

**ЛАБОРАТОРНЫЙ ПРАКТИКУМ**

**по дисциплине**

**“Цифровые и микропроцессорные устройства”**

**часть вторая**

# 1. СИСТЕМА КОМАНД МИКРОПРОЦЕССОРА i8080 (KP580BM80A, KP580ИК80A)

## 1.1. ПРОГРАММНАЯ МОДЕЛЬ МИКРОПРОЦЕССОРА

Для лучшего понимания особенностей системы команд микропроцессора (МП) i8080 (KP580BM80A, KP580ИК80A) воспользуемся его **программной моделью**. Такая модель содержит только **программно-доступные узлы** микропроцессора. Она представлена на рис. 1.1. Знания программной (регистровой) модели вместе со знанием системы команд вполне достаточно для составления прикладных программ.

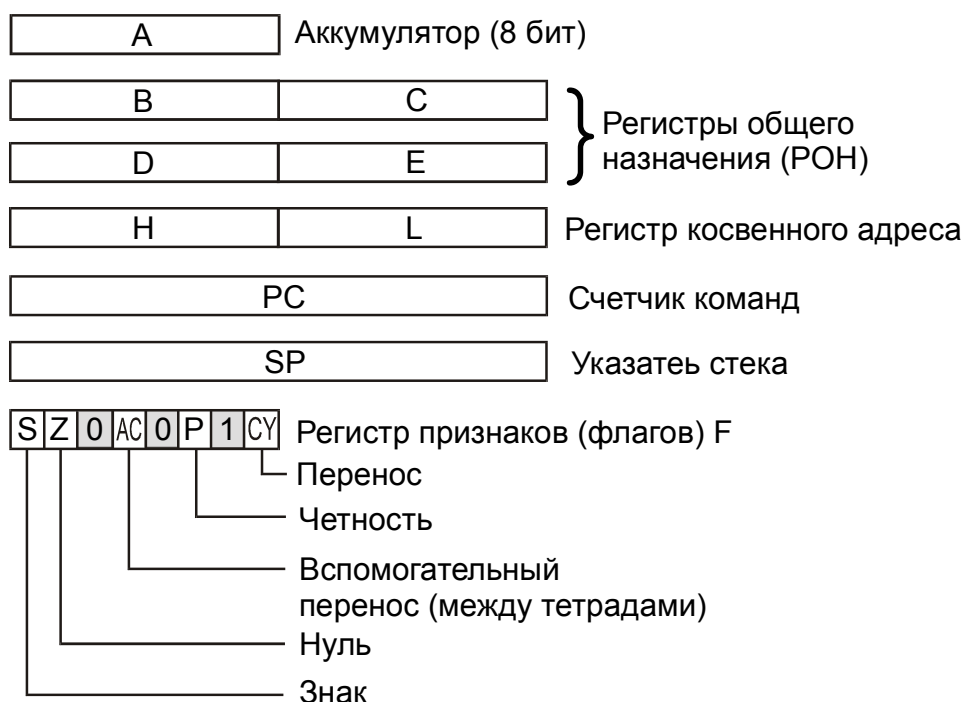


Рис. 1. Программная модель МП i8080.

С точки зрения программиста МП i8080 содержит 8-разрядный аккумулятор (основной регистр процессора, в который заносится один из операндов и в котором сохраняется результат операции). 8-разрядные регистры общего назначения B, C, D, E образуют 16-разрядные пары BC и DE. Пара 8-разрядных регистров H и L образует регистр косвенного адреса HL. Ячейка памяти, адрес которой определяется содержимым пары HL, обозначается M. Программно доступны по чтению и записи 16-разрядные счетчик команд PC и указатель стека SP. По сигналу сброса содержимое PC обнуляется. Указатель стека перед использованием должен быть предварительно проинициализирован командой LXI SP, addr16 (addr16 - некоторый адрес), что позволяет использовать для стека любую область адресного пространства ОЗУ. Регистр признаков (флагов) F содержит следующие флаги: CY (или C) - флаг переноса. Устанавливается в 1 при наличии переноса из старшего бита результата. При вычитании этот флаг становится флагом заема;

S - флаг знака; совпадает со значением старшего бита результата;  
 Z - флаг нуля; устанавливается при нулевом результате;  
 P - флаг четности; устанавливается при наличии в результате четного числа единиц;  
 AC (или A) - флаг вспомогательного переноса; устанавливается при переносе из младшей тетрады в старшую.

## 1.2. РЕЖИМЫ АДРЕСАЦИИ

В поле операнда всех команд каким-либо образом определяются данные, участвующие в операции - операнды. Способ определения операнда называется **режимом адресации**. В МП i8080 используются следующие режимы адресации:

**Прямая адресация** - в поле операнда содержится полный адрес (addr16). Этому обычно соответствует обозначение D (direct) в составе мнемоники команды.

**Непосредственная адресация** - операндом является второй байт самой команды (data8). Этому соответствует обозначение I (immediate) в мнемонике команды.

**Регистровая адресация** - операндом является содержимое регистра, указываемого в команде.

**Неявная адресация** - при этом подразумевается, что операнд находится в определенном регистре и его специально адресовать не нужно.

**Косвенная адресация** - адрес операнда находится в регистре (как правило это пара HL).

## 1.3. КОМАНДЫ МП i8080

Базовая система команд МП i8080 включает 78 простых команд. Многие базовые команды порождают несколько различных кодов операций, поэтому общее их число составляет 244 (см. п.1.4). Команды принято классифицировать на несколько групп, объединяющих команды по их функциональному назначению. Можно выделить группу команд пересылки, группу арифметических операций, группу команд логических операций, группу команд ветвления программ и передачи управления, а также группу команд управления стеком, вводом-выводом и состоянием МП. Эти группы команд приведены ниже. В скобках рядом с мнемоникой команды указано число тактов, необходимых для выполнения команды.

### Группа команд пересылки

В процессе их выполнения признаки состояния (флаги) не изменяются.

Мнемоника (число тактов)	Английское обозн.	Русское обозначение	Операция
<b>MOV R2,R1</b> (5)	(MOVE)	передать из регистра в регистр	(R1)→(R2)
<b>MOV R,M</b> (7)	(MOVE)	передать из памяти в регистр	((HL))→(R)

<b>MOV M,R</b> (7)	(MOVe)	передать из регистра в память	(R)→((HL))
<b>MVI R,data8</b> (7)	(MoVe Immediate)	передать непосредственно	data8→(R)
<b>MVI M,data8</b> (10)	(MoVe Immediate)	передать непосредственно	data8→((HL))
<b>LXI Rp,dat16</b> (10)	(Load register pair (X) Immediate)	загрузить регистровую пару непосредственно	байт2→(С или Е) байт3→(В или D)
<b>LDA addr16</b> (13)	(Load Accumulator Direct)	загрузить аккумулятор прямо	(байт по адресу addr) →(A)
<b>STA addr16</b> (13)	(Store Accumulator Direct)	запомнить аккумулятор прямо	(A)→(по адресу addr)
<b>LHLD addr16</b> (10)	Load HL Direct	загрузить регистровую пару HL прямо	(байт по адресу addr) →(L); (байт по адресу addr+1)→(H)
<b>SHLD addr16</b> (10)	Store HL Direct	запомнить регистровую пару HL прямо	(байт по адресу addr) ←(L); (байт по адресу addr+1)←(H)
<b>LDAX Rp</b> (7)	Load A indirect	загрузить аккумулятор косвенно	((Rp)) →(A)
<b>STAX Rp</b> (7)	Store A indirect	запомнить аккумулятор косвенно	(A) →((Rp))
<b>XCHG</b> (4)	eXCHanGe	обмен содержимого HL и DE	(HL) ↔ (DE)
<b>XTHL</b> (18)	eXchange stack Top with HL	обмен содержимого стека и HL	((SP)) →(L); ((SP)+1) →(H)
<b>SPHL</b> (5)	move HL toSP	загрузить в указатель стека содержимое HL	(HL) →(SP)

Примечание: В качестве R,R1,R2 могут выступать регистры A, B, C, D, E, H, L; в качестве Rp - регистровые пары BC, DE (указывается только B, C соответственно);

data8 - байт данных, dat16 - 16-разрядное слово данных (2 байта); addr16 - 16-разрядный адрес (2 байта, первым идет младший байт, вторым старший); ((HL)) или ((Rp))- содержимое ячейки памяти, адрес которой указан в HL или другой регистровой паре Rp.

### Группа команд арифметических операций

Операции воздействуют на регистр флагов (см. примечание). Вычитание

производится с дополнительным кодом, и появление CY=1 говорит о возникновении заема (borrow)

<b>ADD R</b> (4)	Add register	Сложить содержимое аккумулятора и регистра	$(A)+(R) \rightarrow (A)$
<b>ADD M</b> (7)	Add memory	Сложить содержимое аккумулятора и ячейки памяти	$(A)+((HL)) \rightarrow (A)$
<b>ADI data8</b> (7)	Add Immediate	Сложить непосредственно	$(A)+data8 \rightarrow (A)$
<b>ADC R</b> (7)	Add register with carry	Сложить с содержимым регистра и переносом	$(A)+(R)+(CY) \rightarrow (A)$
<b>ADC M</b> (7)	Add memory with carry	Сложить с содержимым ячейки памяти и переносом	$(A)+((HL))+(CY) \rightarrow (A)$
<b>ACI data8</b> (7)	Add with Carry Immediate	Сложить с переносом непосредственно	$(A)+data8+(CY) \rightarrow (A)$
<b>SUB R</b> (4)	SUBtract register	Вычесть содержимое регистра	$(A)-(R) \rightarrow (A)$
<b>SUB M</b> (7)	SUBtract memory	Вычесть содержимое ячейки памяти	$(A)-((HL)) \rightarrow (A)$
<b>SUI data8</b> (7)	Subtract Immediate	Вычесть непосредственно	$(A)-data8 \rightarrow (A)$
<b>SBB R</b> (4)	SuBtract register with Borrow	Вычесть содержимое регистра с заемом	$(A)-(R)-(CY) \rightarrow (A)$
<b>SBB M</b> (7)	SuBtract memory with Borrow	Вычесть содержимое ячейки памяти с заемом	$(A)-((HL))-(CY) \rightarrow (A)$
<b>SBI data8</b> (7)	Suttract with Borrow Immediate	Вычесть с заемом непосредственно	$(A)-data8-(CY) \rightarrow (A)$
<b>INR R</b> (5)	Increment Register	Инкрементировать регистр	$(R)+1 \rightarrow (R)$
<b>INR M</b> (10)	INcRement memory	Инкрементировать содержимое ячейки памяти	$((HL))+1 \rightarrow ((HL))$
<b>DCR R</b> (5)	DeCrement Register	Декрементировать регистр	$(R)-1 \rightarrow (R)$
<b>DCR M</b> (10)	DeCReament memory	Декрементировать содержимое ячейки памяти	$((HL))-1 \rightarrow ((HL))$
<b>INX Rp'</b> (5)	INcrement register pair	Инкрементировать содержимое пары регистров	$(Rp')+1 \rightarrow (Rp)$
<b>DCX Rp'</b> (5)	DeCrement register pair	Декрементировать содержимое пары регистров	$(Rp')-1 \rightarrow (Rp)$

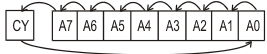
		ров	
<b>DAD Rp'</b> (10)	Doubled ADdition	Сложить содержимое пар регистров Rp и HL	$(Rp')+(HL) \rightarrow (HL)$
<b>DAA</b> (4)	Decimal Adjust Accumulator	Десятичная коррекция аккумулятора	Если младшая тетрада $A > 9$ то $A=A+6$ , $AC=1$ , далее если старшая тетрада $A > 9$ то $A=A+96$ , $CY=1$ .

Примечание: Rp' - регистровые пары BC, DE, HL или SP. В мнемонике команды указывают только старший регистр пары. Команды **INX** и **DCX** не влияют на регистр флагов; команда **DAD** действует только на флаг CY; команды **INR** и **DCR** воздействуют на все флаги, кроме CY.

### Группа команд логических операций

Операции воздействуют на некоторые биты регистра флагов.

<b>ANA R</b> (4)	ANd Accumulator and register	Логическое И аккумулятора и содержимого регистра	$(A) \wedge (R) \rightarrow (A)$ ; $CY=0$ ; $AC=A(3)$
<b>ANA M</b> (7)	ANd Accumulator and memory	Логическое И аккумулятора и содержимого ячейки памяти	$(A) \wedge ((HL)) \rightarrow (A)$ ; $CY=0$ ; $AC=A(3)$
<b>ANI data8</b> (7)	ANd Immediate	Логическое И аккумулятора и непоср. операнда	$(A) \wedge data8 \rightarrow (A)$ ; $CY=0$ ; $AC=A(3)$
<b>XRA R</b> (4)	eXclusive oR Accumulator and register	Исключающее ИЛИ аккумулятора с регистром	$(A) \oplus (R) \rightarrow (A)$ ; $CY=AC=0$
<b>XRA M</b> (7)	eXclusive oR Accumulator and memory	Исключающее ИЛИ аккум. с содерж. ячейки памяти	$(A) \oplus ((HL)) \rightarrow (A)$ ; $CY=AC=0$
<b>XRI data8</b> (7)	eXclusive oR Immediate	Исключающее ИЛИ аккумулятора с непоср. операндом	$(A) \oplus data8 \rightarrow (A)$ ; $CY=AC=0$
<b>ORA R</b> (4)	OR Accumulator and register	Логическое ИЛИ аккумулятора с регистром	$(A) \vee (R) \rightarrow (A)$ ; $CY=AC=0$
<b>ORA M</b> (7)	OR Accumulator and memory	Логическое ИЛИ аккумулятора с содержимым ячейки памяти	$(A) \vee ((HL)) \rightarrow (A)$ ; $CY=AC=0$
<b>ORI data8</b> (7)	OR Immediate	Логическое ИЛИ аккум. с непосредств.. операндом.	$(A) \vee data8 \rightarrow (A)$ ; $CY=AC=0$

<b>CMP R</b> (4)	CoMPare register	Сравнить с регистром	(A)-(R); при этом по окончании операции исходное содержимое аккумулятора сохраняется. Формируются все флаги.
<b>CMP M</b> (7)	CoMPare memory	Сравнить с ячейкой памяти	То же, что и CMP R, но вместо содержимого регистра - содержимое ячейки памяти по адресу (HL)
<b>CPI data8</b> (7)	ComPare Immediate	Сравнить непосредственно	То же, только сравнение со значением второго байта команды (data8)
<b>RAL</b> (4)	RotAte Left	Циклический сдвиг влево	
<b>RAR</b> (4)	RotAte Right	Циклический сдвиг вправо	
<b>RLC</b> (4)	Rotate Left through Carry	Циклический сдвиг влево через перенос	
<b>RRC</b> (4)	Rotate Right through Carry	Циклический сдвиг вправо через перенос	
<b>CMA</b> (4)	CoMplement Accumulator	Инверсия аккумулятора	$(\bar{A}) \rightarrow (A)$
<b>CMC</b> (4)	CoMplement Carry	Инверсия флага переноса	$(\bar{CY}) \rightarrow (CY)$
<b>STC</b> (4)	SeT Carry	Установить флаг переноса	$CY = 1$

Примечание: Команда **CMA** не влияет на флаги, команды **CMC** и **STC** влияют только на флаг CY.

### Группа команд передачи управления

Все флаги сохраняют свои значения

<b>JMP addr16</b>	JuMP	Перейти безусловно	addr16 $\rightarrow$ (PC)
-------------------	------	--------------------	---------------------------

(10)			
<b>J(...)</b> (10)	conditional jump	Переход по условию (...)	Если условие выполняется, то addr16→(PC), иначе PC+1→PC
<b>JC addr16</b>	jump on carry	Переход по переполнению	
<b>JNC addr16</b>	jump on no carry	-“- по отсутствию переполнения	
<b>JZ addr16</b>	jump on zero	-“- по нулевому результату	
<b>JNZ addr16</b>	jump on no zero	-“- по ненулевому результату	
<b>JP addr16</b>	jump on positive	-“- по положительному результату	
<b>JM addr16</b>	jump on minus	-“- по отрицательному результату	
<b>JPE addr16</b>	jump on parity even	-“- по четному результату	
<b>JPO addr16</b>	jump on parity odd	-“- по нечетному результату	
<b>CALL addr16</b> (17)	call	Вызвать безусловно	(PCh) →((SP)-1); (PCl) →((SP)-2); (SP)-2→(SP) addr16→(PC)
<b>C(...)</b> (17)	conditional call	Вызов по условию	Если условие выполняется, то (PCh)→((SP)-1); (PCl) →((SP)-2); (SP)-2→(SP)
<b>CC addr16</b>	call on carry	Вызов по переполнению	
<b>CNC addr16</b>	call on no carry	-“- по отсутствию переполнения	
<b>CZ addr16</b>	call on zero	-“- по нулевому результату	(SP)-2→(SP)
<b>CNZ addr16</b>	call on no zero	-“- по ненулевому результату	
<b>CP addr16</b>	call on positive	-“- по положительному результату	addr16→(PC);
<b>CM addr16</b>	call on minus	-“- по отрицательному результату иначе PC+1→PC	иначе PC+1→PC
<b>CPE addr16</b>	call on parity even	-“- по четному результату	addr16→(PC);
<b>CPO addr16</b>	call on parity odd	-“- по нечетному результату	иначе PC+1→PC



<b>RET</b> (10)	RETurn	Безусловный возврат	$((SP)) \rightarrow (PCl);$ $((SP)+1) \rightarrow (PCh);$ $(SP)+2 \rightarrow (SP)$
<b>R(...)</b> (11)	return on condition	Возврат по условию	Если условие вы- полняется, то $((SP)) \rightarrow (PCl);$ $((SP)+1) \rightarrow (PCh);$ $(SP)+2 \rightarrow (SP);$ иначе $PC+1 \rightarrow PC$
<b>RC</b>	return on carry	Возврат по переполнению	
<b>RNC</b>	return on no carry	-“- по отсутствию переполнения	
<b>RZ</b>	return on zero	-“- по нулевому результату	
<b>RNZ</b>	return on no zero	-“- по ненулевому результату	
<b>RP</b>	return on positive	-“- по положительному результату	
<b>RM</b>	return on minus	-“- по отрицательному результату	
<b>RPE</b>	return on parity even	-“- по четному результату	
<b>RPO</b>	return on parity odd	-“- по нечетному результату	
<b>RST n</b> (n=1-7) (11)	restart	Повторный пуск	$(PCh) \rightarrow$ $((SP)-1);$ $(PCl) \rightarrow ((SP)-2);$ $(SP)-2 \rightarrow (SP);$ $8*n \rightarrow (PC)$
<b>PCHL</b> (5)	HL to PC	Безусловный переход по косвенному адресу.	$(HL) \rightarrow (PC)$

Примечание: PCh, PCl - соответственно старший и младший байты счетчика команд.

## Группа команд управления стеком, вводом-выводом и состоянием МП

Все флаги сохраняют свои значения

<b>PUSH Rp</b> (11)	push	Поместить в стек	$R_{ph} \rightarrow ((SP)-1);$ $R_{pl} \rightarrow ((SP)-2);$ $(SP)-2 \rightarrow (SP)$
<b>PUSH PSW</b> (11)	push processor status word	Поместить слово состоя- ния процессора в стек	$(A) \rightarrow ((SP)-1);$ $(F) \rightarrow ((SP)-2);$ $(SP)-2 \rightarrow (SP)$
<b>POP Rp</b> (10)	pop	Вытолкнуть из стека	$((SP)) \rightarrow (R_{pl});$ $((SP)+1) \rightarrow (R_{ph})$ $(SP)+2 \rightarrow (SP)$
<b>POP PSW</b> (10)	pop processor status word	Вытолкнуть слово со- стояния процессора из стека	$((SP)) \rightarrow (F);$ $((SP)+1) \rightarrow (A)$ $(SP)+2 \rightarrow (SP)$
<b>IN port</b> (10)	INput	Ввод данных из порта	$(port) \rightarrow (A)$
<b>OUT port</b> (10)	OUTput	Вывод данных в порт	$(A) \rightarrow (port)$
<b>EI</b> (4)	Enable Interrupts	Разрешить прерывания	
<b>DI</b> (4)	Disable Interrupts	Запретить прерывания	
<b>HLT</b> (7)	HaLT	Стоп	
<b>NOP</b> (4)	No OPeration	Пустая команда	

Примечание: Rp - регистровые пары BC, DE, HL или PSW (=F,A); Rpl и Rph - соответственно младший и старший регистр пары; в команде указывается только старший регистр (B, D, H) или PSW; port - адрес порта устройства ввода-вывода.

#### 1.4. ТАБЛИЦА КОДОВ КОМАНД МП i8080 (KP580BM80A, KP580IK80A)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LXI B,&	STAX B	INX B	INR B	DCR B	MVI B,#	RLC	-	DAD B	LDAX B	DCX B	INR C	DCR C	MVI C,#	RRC
1	-	LXI D,&	STAX D	INX D	INR D	DCR D	MVI D,#	RAL	-	DAD D	LDAX D	DCX D	INR E	DCR E	MVI E,#	RAR
2	-	LXI H,&	SHLD *	INX H	INR H	DCR H	MVI H,#	DAA	-	DAD H	LHLD *	DCX H	INR L	DCR L	MVI L,#	CMA
3	-	LXI SP,&	STA *	INX SP	INR M	DCR M	MVI M,#	STC	-	DAD SP	LDA *	DCX SP	INR A	DCR A	MVI A,#	CMC
4	MOV B,B	MOV B,C	MOV B,D	MOV B,E	MOV B,H	MOV B,L	MOV B,M	MOV B,A	MOV C,B	MOV C,C	MOV C,D	MOV C,E	MOV C,H	MOV C,L	MOV C,M	MOV C,A
5	MOV D,B	MOV D,C	MOV D,D	MOV D,E	MOV D,H	MOV D,L	MOV D,M	MOV D,A	MOV E,B	MOV E,C	MOV E,D	MOV E,E	MOV E,H	MOV E,L	MOV E,M	MOV E,A
6	MOV H,B	MOV H,C	MOV H,D	MOV H,E	MOV H,H	MOV H,L	MOV H,M	MOV H,A	MOV L,B	MOV L,C	MOV L,D	MOV L,E	MOV L,H	MOV L,L	MOV L,M	MOV L,A
7	MOV M,B	MOV M,C	MOV M,D	MOV M,E	MOV M,H	MOV M,L	HLT	MOV M,A	MOV A,B	MOV A,C	MOV A,D	MOV A,E	MOV A,H	MOV A,L	MOV A,M	MOV A,A
8	ADD B	ADD C	ADD D	ADD E	ADD H	ADD L	ADD M	ADD A	ADC B	ADC C	ADC D	ADC E	ADC H	ADC L	ADC M	ADC A
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB M	SUB A	SBB B	SBB C	SBB D	SBB E	SBB H	SBB L	SBB M	SBB A
A	ANA B	ANA C	ANA D	ANA E	ANA H	ANA L	ANA M	ANA A	XRA B	XRA C	XRA D	XRA E	XRA H	XRA L	XRA M	XRA A
B	ORA B	ORA C	ORA D	ORA E	ORA H	ORA L	ORA M	ORA A	CMP B	CMP C	CMP D	CMP E	CMP H	CMP L	CMP M	CMP A
C	RNZ	POP B	JNZ *	JMP *	CNZ *	PUSH B	ADI #	RST 0	RZ	RET	JZ *	-	CZ *	CALL *	ACI #	RST 1
D	RNC	POP D	JNC *	OUT N	CNC *	PUSH D	SUI #	RST 2	RC	-	JC *	IN N	CC *	-	SBI #	RST 3
E	RPO	POP H	JPO *	XTHL	CPO *	PUSH D	ANI #	RST 4	RPE	PCHL	JPE *	XCHG	CPE *	-	XRI #	RST 5
F	RP	POP PSW	JP *	DI	CP *	PUSH PSW	ORI #	RST 6	RM	SPHL	JM *	EI	CM *	-	CPI *	RST 7

N - адрес порта ввода-вывода; & - двухбайтовый операнд (data16);

\* - двухбайтовый операнд (addr16);# - однобайтовый операнд (data 8)

Пример: команда STAX D имеет код операции 12h;

Код операции CAh соответствует команде JZ addr.

## **2. ЭМУЛЯТОР МИКРОПРОЦЕССОРНОЙ СИСТЕМЫ 8080SDE**

### **2.1. НАЗНАЧЕНИЕ И ОБЩИЕ ХАРАКТЕРИСТИКИ ПРОГРАММЫ**

Программа 8080SDE эмулирует микропроцессорную систему, включающую МП i8080 и имеющую объем памяти программ пользователя 64 К (адреса с 0000H по FFFFH).

Программа имеет 3 основные функции:

- Редактирование ассемблерных и объектных файлов;
- Трансляция файлов на языке АССЕМБЛЕРА в объектный код и дизассемблирование объектных файлов;
- Эмуляция выполнения объектных файлов.

Эмулируется микропроцессорная система с разделенными шинами адресов памяти и устройств ввода-вывода. Устройствами ввода-вывода являются:

- Регистр переключателей (8 кнопок с независимой фиксацией, подключены к порту ввода);
- Панель светодиодов (LED display) - 8 светодиодов, подключенных к порту вывода;
- Панель семисегментных индикаторов (LCD display) - 4 индикатора, подключены к портам вывода;
- Графический планшет (имитирует работу цветного графического дисплея)
- Матричный дисплей (MUX display) - имитирует работу полноцветной светодиодной панели с организацией 32 столбца × 8 строк.
- Устройство Active Wire USB (для функционирования необходимо наличие этого внешнего устройства, подключенного к USB порту компьютера. В данной конфигурации не используется).

### **2.2. УПРАВЛЕНИЕ ПРОГРАММОЙ**

Вид главного окна программы после ее запуска показан на рис. 2.1. Основными его элементами являются строка меню 1, окно просмотра значений ячеек памяти (HEX-viewer) 2, строка редактирования ассемблерных команд (Edit) 3, окно просмотра текста программы на языке Ассемблера (ASM-viewer) 4, окно Дизассемблера (Dis-Assembly) 5, а также кнопки включения режимов программы 6 (ASSEMBLE (трансляция), EMULATE (эмуляция), DISASSEMBLE (дизассемблирование), CANCEL (отмена), EXIT (выход из программы)).

#### **2.2.1. Редактирование**

Поместить курсор в ячейку окна Assembly или окна просмотра HEX-файлов и щелкнуть левой клавишей мыши. Если значение уже находится в ячейке, оно будет вызвано для редактирования. Нажим <ENTER> вводит значение в ячейку и продвигает курсор к следующей ячейке. Перемещение по ячейкам этих окон можно осуществлять с помощью клавиш ↑ (вверх), ↓ (вниз), F11 (влево), F12 (вправо).

#### **2.2.2. Команды меню File (файл)**

## Сохранение ассемблерных (A80) и объектных (HEX) файлов (Save)

Щелчок на File -> Save вызывает стандартное окно диалога записи/чтения файла Windows. Вы можете изменять имя каталога для сохранения/загрузки. Также может быть выбран тип файла (.A80 для файлов на языке Ассемблера во внутреннем формате системы; .HEX для объектных файлов). Файлы на языке Ассемблера - последовательности строк, а HEX-файлы являются двоичными. При сохранении или загрузке в режиме REQUESTERS ENABLED (разрешены запросы) выводится запрос о НАЧАЛЕ / КОНЦЕ. Для файлов АССЕМБЛЕРА, НАЧАЛО / КОНЕЦ - это ДЕСЯТИЧНЫЕ НОМЕРА СТРОКИ. Для объектных файлов НАЧАЛО / КОНЕЦ - это шестнадцатеричные АДРЕСА.

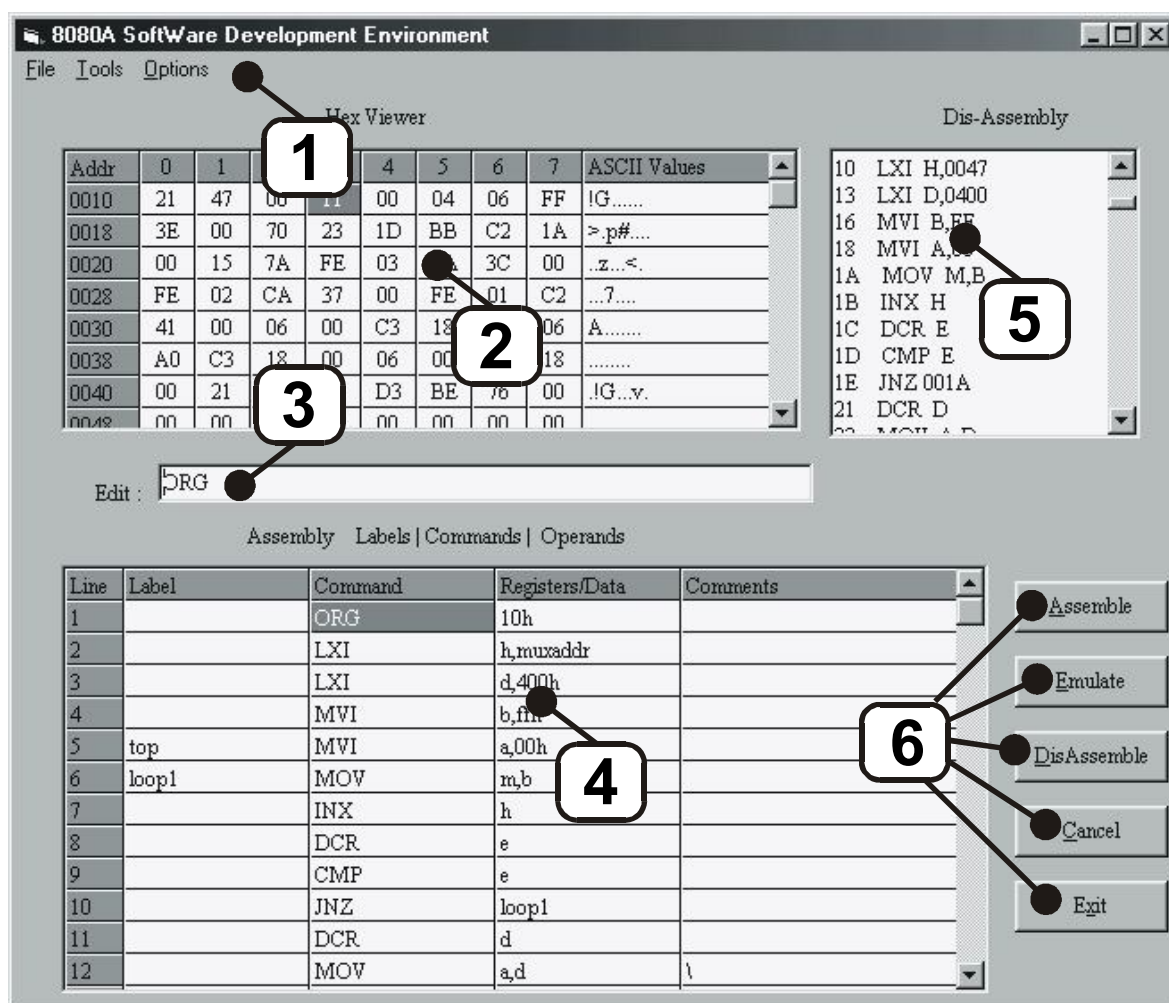


Рис.2.1. Вид основного окна программы 8080SDE.

## Загрузка ассемблерных (A80) и объектных (HEX) файлов (Load)

File -> Load. Процесс, подобный действиям при сохранении файлов. Могут быть запрошены номера строк для ассемблерных и начальный/конечный адреса для объектных (HEX) файлов.

## Printing - Печать

Вывод на печать содержимого окна ассемблерной программы. Если не указан номер начальной строки, по умолчанию выбирается строка 1. Если не указана конечная строка, будет печататься все содержимое окна ассемблерной программы, пока не встретится директива END.

### 2.2.3. Команды меню Tools (инструменты)

Команды могут быть выполнены как с помощью меню, так и нажатием служебных клавиш F1-F9 (указаны в скобках).

**Insert (F2)/ Delete ASM Lines (F3) - Вставка / удаление строк ассемблерной программы**

Строка ПОД курсором будет удалена; новая строка будет вставлена также ПОД курсором.

**Search (F8)/ Clear Search (F1)- Поиск / Очистка объекта поиска**

Поиск значения высветит все встречающиеся значения БАЙТА в HEX-вьюере (окне просмотра дампа памяти). Очистка объекта поиска очистит (обнулит) ВСЕ найденные байты.

**Zero (F5)/ Fill HEX Display (F6)- Обнуление / Заполнение памяти константой**

При обнулении обнуляется вся область памяти по адресам 0000H-FFFFH. Заполнение константой позволяет заполнить область памяти по Вашему выбору некоторой константой.

**Clear ASM viewer (F9) - стереть ассемблерную программу**

Полностью стирает содержимое окна просмотра ассемблерного кода. Требуется осторожного использования, так как отменить это действие невозможно.

**Clear Dis-Assembler Display (F7) - Очистка окна дизассемблера.**

Очищает окно дизассемблера.

**Refresh HEX Display (F4) - Обновление окна просмотра памяти.**

В течение трансляции и эмуляции программы, происходит/может происходить запись в память. Для просмотра изменений в памяти необходимо обновить содержимое окна просмотра памяти.

### 2.2.4. Команды меню Options (режимы)

Это меню позволяет включить/выключить режим **Requesters** (запросы). Включение этого режима программа запрашивает начальный/конечный адрес или строки при операциях загрузки/сохранения, печати, трансляции или дизассемблирования. При отключенном режиме используются установки по умолчанию, а именно при загрузка/сохранение ассемблерных программ происходит в интервале строк 0-3000 (или до директивы END), загрузка/сохранение двоичных файлов происходит в интервале адресов от 0000h до FFFFh, при дизассемблировании дизассемблируются 256 байт начиная с нулевого адреса (или продолжается дизассемблирование следующих 256 байт). Ассемблер обрабатывает по умолчанию диапазон строк 0-3000 или до директивы END. Другой пункт меню - установка величины экрана

(**Screen Size**). Позволяет настроить программу на заданное разрешение экрана видеотерминала компьютера.

### 2.2.5. Встроенный Ассемблер

Ассемблер - это программа, преобразующая последовательность мнемонических инструкций (текст программы на языке Ассемблера) в последовательность машинных кодов команд и двоичных данных, называемую объектным кодом. Именно эти коды считываются из памяти и выполняются микропроцессором. Окно текста программы на языке Ассемблера представляет из себя таблицу, содержащую 4 столбца, соответствующих полю меток, полю мнемоник команд (директив), полю операндов и полю комментариев. Каждая команда занимает одну строку таблицы. В программе 8080SDE строки пронумерованы. Встроенный Ассемблер - очень простой двухпроходный ассемблер. Он распознает следующие обозначения и директивы:

#### Обозначения:

- H, h - числа в шестнадцатеричном представлении;
- Q, q - числа в восьмеричном представлении;
- B, b - числа в двоичном представлении;
- по умолчанию - числа в десятичном представлении.

#### Директивы:

В отличие от операторов Ассемблера, включающих мнемоники команд и операнды, директивы Ассемблера не порождают машинных кодов. Директивы, называемые также псевдокомандами, - это инструкции по выполнению трансляции программы. Директивы программы 8080SDE:

ORG - Начало кода. В коде может использоваться несколько директив ORG. Например, директива ORG 10h устанавливает адрес следующей команды 10h.

EQU - ставит в соответствие число символу. Например testsym EQU 1000q приравнивает символ testsym восьмеричному значению 1000

DB - Определяет байт(ы) в поле операндов, разделенные запятыми.

DW - Определяет 16-разрядные слова в поле операндов, разделенные запятыми.

DS - Определяет строку значений. Например, testsym DS 8,33 определяет 8 байтов, имеющих десятичное значение 33 каждый.

DT - Определить текст. Например, testsym DT 'Hello' определяет текстовую строку.

END - Конец ассемблерного кода. Все, что находится после директивы END, Ассемблером не обрабатывается.

#### Symbols - Метки

Метки можно поставить в любой строке. Следующие метки недопустимы:

- зарезервированные мнемоники команд;
- зарезервированные директивы;
- обозначения регистров процессора;
- числа.

Метки могут содержать символы, числа и символы пунктуации. Однако, возможны некоторые побочные эффекты, если нарушен стандартный синтаксис меток. Двоеточие в конце метки необязательно.

### Комментарии

Комментарии могут быть на любой строке и не включаются в объектный код. Точка с запятой не необходима в начале комментария, в отличие от требований стандартного Ассемблера.

### 2.2.6. Трансляция

Трансляция файла в объектный код происходит по нажатию кнопки ASSEMBLE в главном окне программы. По окончании трансляции возникает окно сообщений транслятора (рис.2.2). Сообщения, не содержащие кодов ошибок, говорят об успешной трансляции. Для просмотра кода оттранслированной программы в памяти необходимо освежить содержимое окна просмотра памяти.

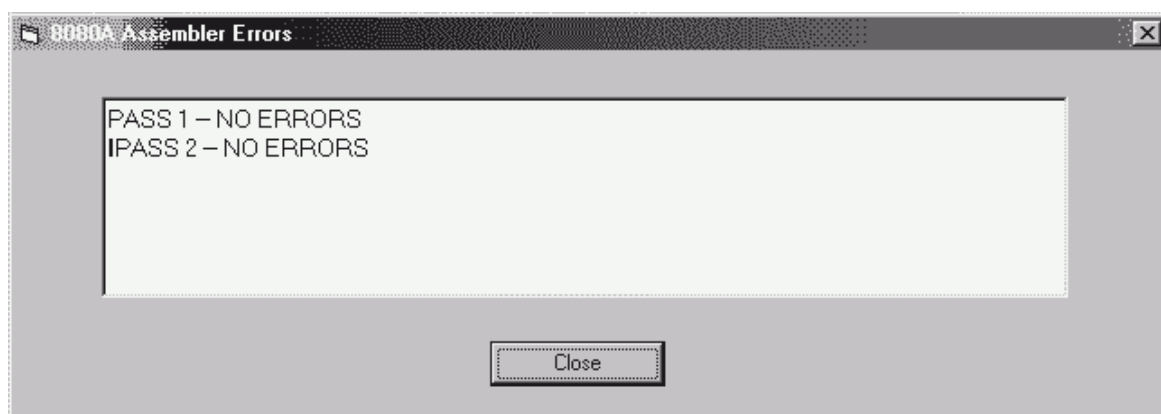


Рис.2.2. Окно сообщений встроенного Ассемблера программы 8080SDE.

### 2.2.7. Эмуляция

Как только ассемблерный файл был успешно оттранслирован, или был загружен HEX-файл, может быть включен режим эмуляции (EMULATE). При этом возникает окно эмулятора (рис.2.3), в котором отображаются текущие значения регистра флагов F (флаги Z, CY, S, P, AC, обозначенные соответственно Z, C, S, P, A), аккумулятора, регистров общего назначения B, C, D, E, H, L, программного счетчика PC, указателя стека SP, мнемоники текущей (подлежащей выполнению) команды, некоторых портов ввода-вывода. Имеется также кнопка аппаратного сброса системы (Reset) и движок управления скоростью прогона программы. Вначале следует загрузить начальный адрес объектного кода в поле PC. Можно загружать также значения и в другие регистры. Нажатие клавиши LOAD загружает текущие значения, отображаемые в окнах регистров, в соответствующие регистры. Выберите основание системы счисления для представления данных (двоичное, восьмеричное, шестнадцатеричное, десятичное). Выберите также режим исполнения (run mode). Возможны следующие режимы.



1) Step (Шаг) - выбирает пошаговый режим. Кнопка START нажимается каждый раз, чтобы выполнить команду.

2) Coarse (Грубо)- грубая подстройка скорости прогона программы.

3) Fine (Точно) - более точная подстройка скорости выполнения программы. В крайнем правом положении движка регулятора программа выполняется с максимальной скоростью.

Кнопка CLOSE используется для закрытия окна эмулятора.

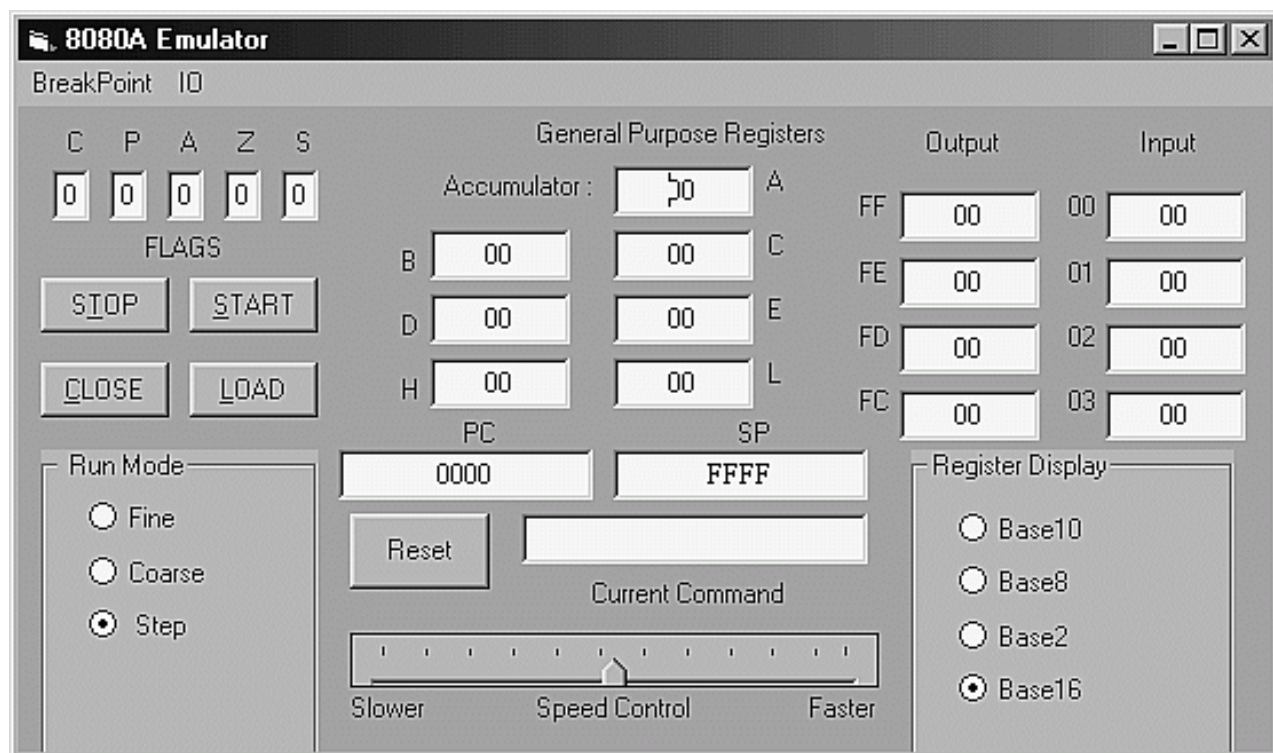


Рис.2.3. Окно эмулятора микропроцессорной системы программы 8080SDE

В меню окна эмуляции доступны следующие режимы:

### Breakpoints - Контрольные точки останова

Вызывает окно установки точек останова (рис.2.4). Можно установить 3 различные контрольные точки останова. В окне WHEN выбирается из списка наблюдаемый регистр или флаг; IS определяет задаваемое условие для сравнения (варианты: LT (less than) - меньше, LE (less or equal) - меньше или равно, EQ (equal) - равно, GE (greater or equal) - больше или равно, GT (greater than) - больше, NE (not equal) - не равно), VALUE соответствует значению, с которым сравнивается содержимое регистра.

Щелчок во флажке слева от полей контрольной точки активизирует эту конкретную контрольную точку. Если установлено несколько контрольных точек, они анализируются по логике ИЛИ.

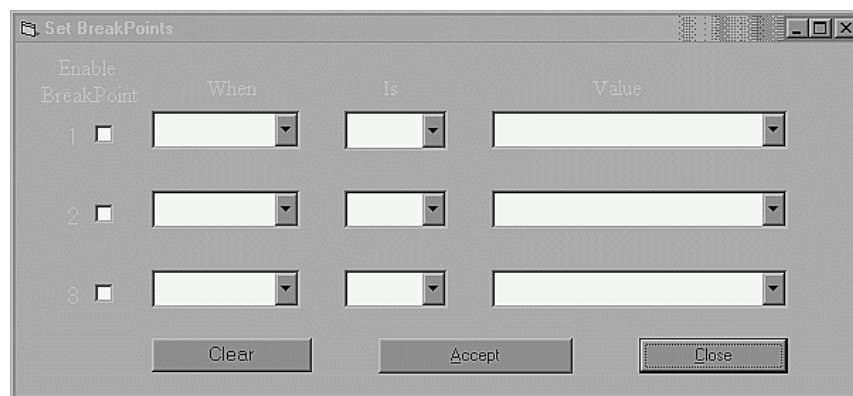


Рис. 2.4. Окно ввода контрольных точек останова программы 8080SDE.

### Ю - Устройства ввода-вывода.

Можно вызвать окна отображения светодиодного индикатора (LED) (рис.2.5), семисегментных индикаторов (LCD) (рис.2.6), матричного дисплея (рис.2.7) или графического планшета (Graphic Tablet). В окне светодиодного индикатора отображаются также кнопки-переключатели входного регистра (рис. 2.5).

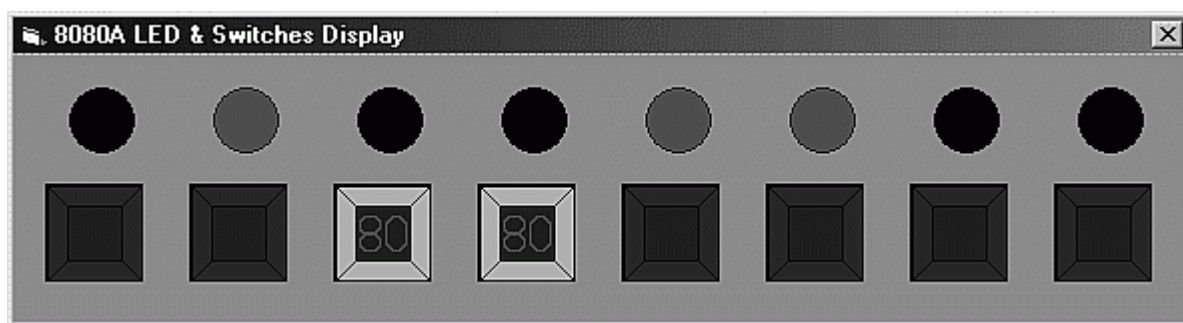


Рис. 2.5. Окно светодиодного индикатора (LED) и кнопочного регистра.

### Графика

Следующие значения и порты управляют планшетом графического вывода (через аккумулятор)

Порт  $210_{10}$  - слово управления - следующие значения здесь выполняют функции:

8 - Изменить цвет

$16_{10}$  - Автоперерисовка - Используется вместе с режимом очистки экрана

$32_{10}$  - очистка экрана

$64_{10}$  - нарисовать от текущего расположения курсора до значений в BC ( $X_2$  - координата) и DE ( $Y_2$  - координата)

$128_{10}$  - Расположить курсор в позиции с координатами, определяемыми значениями в BC ->  $x1$  и DE ->  $y1$

Порт 215 - Интенсивность красного цвета от 0 до  $255_{10}$

Порт 216 - Интенсивность зеленого цвета "-"-"

Порт 217 - Интенсивность синего цвета "-"-"

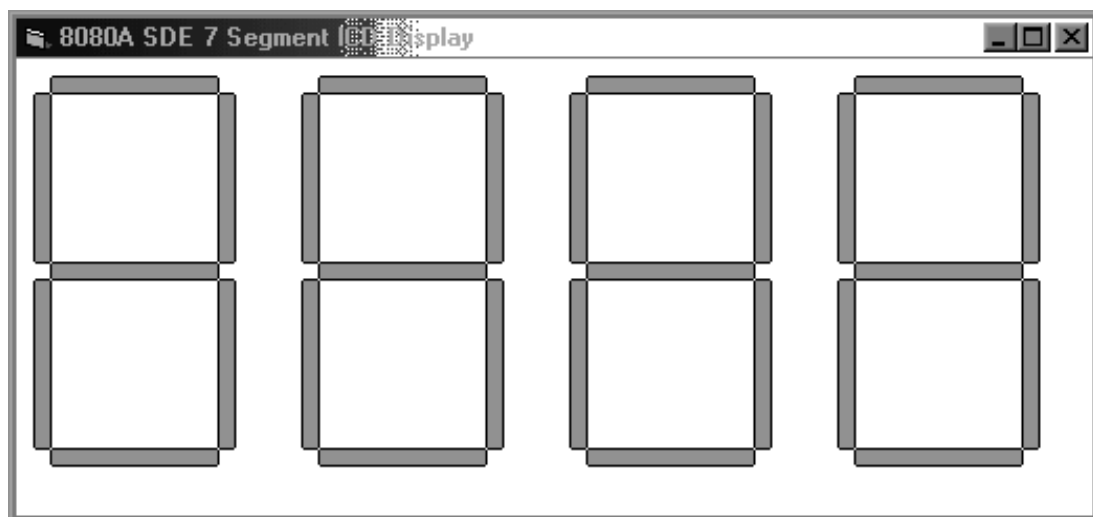


Рис. 2.6. Панель семисегментных индикаторов (LCD).

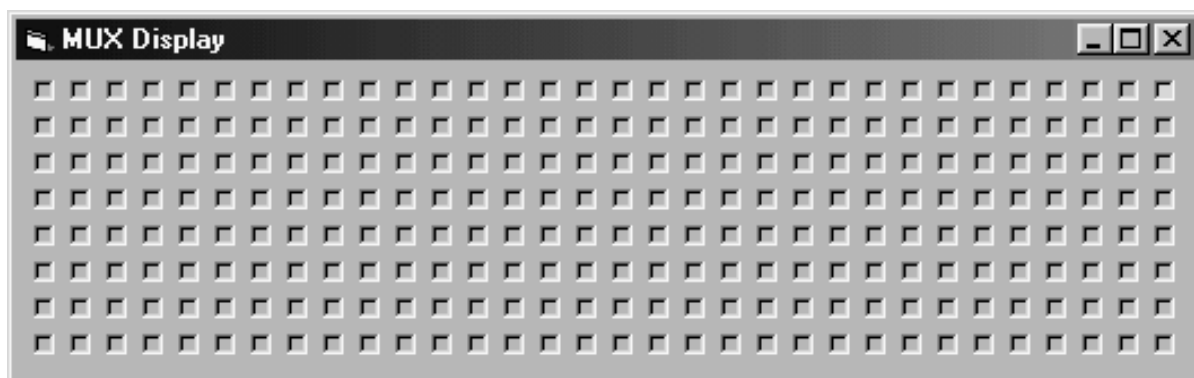


Рис. 2.7. Панель матричного дисплея (MUX).

Расположение графического курсора включает загрузку регистровой пары BC значением  $X_1$  и пары DE значением  $Y_1$ , загрузкой в аккумулятор значения  $128_{10}$  и затем запись в порт  $210_{10}$ . Курсор автоматически модифицируется. Чтобы провести линию, надо загрузить в регистровые пары BC и DE соответственно значения  $X_2$  и  $Y_2$ , загрузить в аккумулятор значение  $64_{10}$  и затем произвести вывод в порт  $210_{10}$ . Убедитесь, что перед этим установили цвета.

### Светодиодная панель

Порт  $220_{10}$ . Посылка значения из аккумулятора в этот порт зажигает соответствующие светодиоды. Самый младший бит соответствует правому светодиоду.

### Панель семисегментных индикаторов

Порт  $230_{10}$ . Чтобы вывести значение, необходимо загрузить в B отображаемое значение в соответствии с приведенной схемой (рис.2.8), а в C - номера зажигаемых индикаторов (1 - 4), устанавливаемые битами младшей тетрады байта (младший бит соответствует правому разряду индикатора, старший - левому). Так, чтобы зажечь все 4 разряда, необходимо занести в C число  $15_{10}$  (0FH).

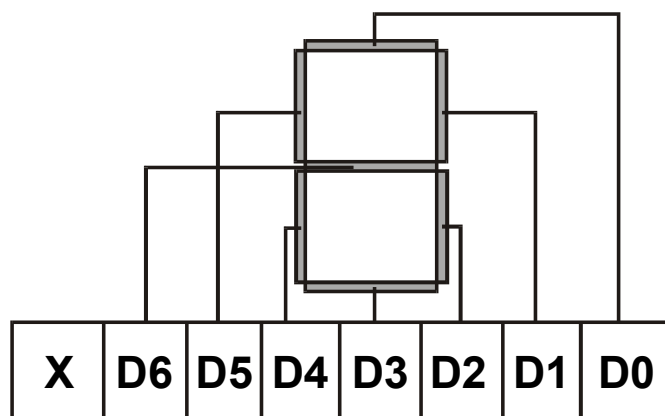


Рис. 2.8. Схема управления сегментами индикатора установкой бит слова в регистре В.

С учетом этой схемы, например, чтобы отобразить число 9, надо установить комбинацию  $2^0 + 2^1 + 2^2 + 2^3 + 2^5 + 2^6 = 1 + 2 + 4 + 8 + 32_{10} + 64_{10} = 111_{10}$

При размещении  $111_{10}$  в регистре В и затем  $1 - 15_{10}$  в регистре С после команды  $\text{OUT } 230_{10}$  ( $\text{OUT } \text{E6H}$ ) зажгутся соответствующие разряды индикатора, показывая цифру 9.

### Регистр переключателей

Порт  $20_{10}$ . ввод из этого порта помещает значения вкл/выкл (1/0) этих переключателей в аккумулятор. Самый младший бит слова состояния этого регистра переключателей соответствует левой стороне панели.

### Матричный дисплей

Устройство эмулирует светодиодную панель  $32 \times 8$  (всего 256 элементов). Порт  $190_{10}$  - строб матричного дисплея. Управление дисплеем формируется в соответствии со сформированным в памяти массивом пользователя из 1024 элементов. При этом элементы 0-255 (00H-FFH) соответствуют зажиганию соответствующих пикселей (нумерация возрастает справа налево и сверху вниз, так что правому крайнему пикселю в верхней строчке соответствует адрес 00H, крайнему левому - 1FH, крайнему правому во второй строке сверху - 20H и т.д). Активация элементов происходит при записи в массив по адресам элементов с номерами от 00H до FFH кода FF по соответствующему адресу. При записи любого другого кода соответствующие элементы дисплея неактивны. Элементы с номерами 100H-1FFH, 200H-2FFH, 300H-3FFH соответственно управляют интенсивностью красного, зеленого и синего цветов каждого из элементов. Запись 00H соответствует нулевой яркости, FFH - максимальной яркости свечения соответствующего цвета. Таким образом, панель является полноцветной. Для активации дисплея после формирования массива записывают в регистровую пару HL адрес начала массива и затем подают команду  $\text{OUT } 190_{10}$ .

**Active Wire USB** (AWUSB, устройство компании Activewire, <http://www.activewireinc.com>, в данной конфигурации не используется).

Чтобы ОТКРЫТЬ устройство AWUSB, произвести вывод в порт  $240_{10}$

Чтобы ЗАКРЫТЬ устройство AWUSB, произвести вывод в порт  $243_{10}$

Чтобы РАЗРЕШИТЬ конфигурацию контактов разъема ввода/вывода на открытом устройстве, регистровая пара ВС должна содержать значение слова конфигурации (например, значение F000H в ВС означает, что контакты 11 - 15 открытого AWUSB устройства работают на вывод, а контакты 0 - 10 - на ввод. Вывод в порт  $242_{10}$ .

Чтобы ЗАПИСАТЬ значение в открытое AWUSB устройство, загрузите ВС значением, затем произведите вывод в порт  $241_{10}$ .

Чтобы ВВЕСТИ значение из открытого AWUSB устройства, произведите ввод из порта 10. Входное значение будет в регистровой паре ВС.

Во всех случаях, если данная операция (открытие, закрытие, разрешение, запись, чтение) успешна, в аккумуляторе будет возвращен 0. Если возникает ошибка - в аккумуляторе будет код  $255_{10}$ .

### **2.2.8. Дизассемблирование**

Нажатие кнопки DISASSEMBLE вызывает дизассемблирование объектного кода и отображение листинга в окне дизассемблера. Если выключен режим Requesters, производится дизассемблирование кода, расположенного по адресам 0000H-00FFH. Последующие нажатия вызывают дизассемблирование следующих порций кода по 256 байт. Какого-либо распознавания текста, таблиц и т.д. дизассемблер не проводит.

### 3. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

#### ЛАБОРАТОРНАЯ РАБОТА № 1

#### ИЗУЧЕНИЕ ПРОГРАММНОЙ МОДЕЛИ ПРОЦЕССОРА i8080, СИСТЕМЫ КОМАНД, ЭМУЛЯТОРА МП СИСТЕМЫ 8080SDE, СОСТАВЛЕНИЕ И ОТЛАДКА ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ КОМАНД ПЕРЕДАЧИ ДАННЫХ

**Цель работы** - изучить программную модель микропроцессора i8080 (KP580BM80A, ИК80А), систему его команд, приобрести навыки работы с программой-симулятором микропроцессорной системы 8080SDE, научиться составлять простейшие программы с использованием команд передачи данных и выполнять их с помощью 8080SDE.

**Теоретическая часть:** ознакомьтесь с содержанием разделов 1.1, 1.2 1.3 в части команд пересылки данных, 1.4 и раздела 2 данного пособия.

**Задания:**

1. Загрузите программу 8080SDE. Ознакомьтесь с видом главного окна, расположением элементов, содержанием главного меню и выпадающих меню главного окна.
2. Включите режим REQUESTERS.
3. Обнулите содержимое памяти с адреса 0000h по адрес FFFFh, затем заполните память какой-либо константой (например B6h), снова обнулите содержимое памяти.
4. Введите в память начиная с адреса 0000h последовательности кодов из таблицы 3.1:

Таблица 3.1

Вариант 1	Вариант 2	Вариант 3	Вариант 4
3A 3F 09 47 21 1F 09 7E 05 CA 14 09 23 BE DA 07 09 C3 08 09 21 FE 09 77	21 40 09 0E 00 11 80 09 1A 13 77 23 0C D6 00 C2 08 09	21 40 09 11 80 09 7E FE 2E CA 12 09 12 23 13 C3 06 09	06 04 11 1F 09 21 2F 09 AF 1A 8E 12 05 C8 23 13 C3 09 09
Вариант 5	Вариант 6	Вариант 7	Вариант 8
3A 3F 09 47 21 1F 09 97 86 23 05 C2 08 09 21 FE 09 77	3A 0F 00 47 21 1F 09 7E 83 5F 23 7E 8A 57 23 05 C2 07 09 EB 22 FE 09	21 00 09 0E 00 11 1F 09 EB 7E EB 13 77 23 0C D6 00 C2 08 00	06 02 11 00 09 21 02 09 37 3E 99 CE 00 96 EB 86 27 77 EB 05 CA 1C 00 13 23 C3 09 00

5. Выполните команду Search HEX value в диапазоне 0000h-00FFh, задав искомый код 09h. Сколько полей высветилось в HEX-вьюере? Выполните команду Clear Search.
6. Дизассемблируйте введенный вами код и запишите полученный текст программы на языке Ассемблера. Обнулите содержимое памяти по адресам 0000h-FFFFh.
7. Введите текст программы на языке Ассемблера из таблицы 3.2, оттранслируйте программу, запишите содержимое ячеек памяти по адресам 0000h-003Fh после трансляции.
8. Перейдите в режим эмуляции, при необходимости обнулите регистры командой Reset и загрузите в регистр PC значение 0000h командой Load, выполните программу в пошаговом режиме до первой команды NOP, наблюдая за изменением содержимого регистров при выполнении команд и занося результаты в таблицу по форме таблицы 3.
9. Выполните команду Refresh HEX Display и изучите произошедшие изменения информации в памяти. По результатам напишите комментарии к тексту программы (последняя колонка таблицы 3.3).

Таблица 3.2

Вариант 1	Вариант 2	Вариант 3	Вариант 4
ORG 00h MVI A,18h MVI C,7Eh MOV B,A MOV E,C LXI D,0037h LXI H,8899h XCHG MOV M,A LHLD 0030h XCHG LDAX D STA 0038h LDA 0033h MVI B,00h MVI C,39h STAX B SHLD 003Ah NOP ORG 0030h DB 32h,00h DB F1h,F2h DB F3h,F4h DB 00h,00h NOP END	ORG 00h MVI A,3Ah MVI C,2Dh MOV B,A MOV E,C LXI D,0037h LXI H,D7AEh XCHG MOV M,A LHLD 0030h XCHG LDAX D STA 0038h LDA 0033h MVI B,00h MVI C,39h STAX B SHLD 003Ah NOP ORG 0030h DB 32h,00h DB 43h,44h DB 45h,46h DB 00h,00h NOP END	ORG 00h MVI A,E5h MVI C,11h MOV B,A MOV E,C LXI D,0037h LXI H,1234h XCHG MOV M,A LHLD 0030h XCHG LDAX D STA 0038h LDA 0033h MVI B,00h MVI C,39h STAX B SHLD 003Ah NOP ORG 0030h DB 32h,00h DB D5h,D6h DB D7h,D8h DB 00h,00h NOP END	ORG 00h MVI A,7Bh MVI C,4Fh MOV B,A MOV E,C LXI D,0037h LXI H,A6B2h XCHG MOV M,A LHLD 0030h XCHG LDAX D STA 0038h LDA 0033h MVI B,00h MVI C,39h STAX B SHLD 003Ah NOP ORG 0030h DB 32h,00h DB 7Ch,7Dh DB 7Eh,7Fh DB 00h,00h NOP END

Таблица 3.3

Адрес	Команда	Коды	Содержимое регистров							Комментарии
			A	B	C	D	E	H	L	

**Форма отчета:**

1. Листинг дизассемблированной программы (задание 6)
2. Содержимое ячеек памяти после трансляции программы (задание 7) и после выполнения программы (задание 9)
3. Таблица хода пошагового выполнения программы по форме таб.3.3 (задания 8,9).

**Контрольные вопросы**

1. Что такое программная модель микропроцессора?
2. Расскажите о программно-доступных регистрах микропроцессора i8080 и их назначении.
3. Сколько команд входит в систему команд МП i8080?
4. На какие группы подразделяется совокупность команд микропроцессора i8080?
5. Какие режимы адресации используются в командах МП i8080, как они отображаются в мнемонике команд? Приведите примеры.
6. Что такое Ассемблер? В чем отличие между операторами и директивами Ассемблера?
7. Какие поля содержатся в строке ассемблерной программы?
8. Эмуляция каких внешних устройств предусмотрена программой 8080SDE?

## ЛАБОРАТОРНАЯ РАБОТА № 2.

### КОМАНДЫ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ МП i8080

**Цель работы:** Изучение команд арифметических операций.

**Краткие теоретические сведения.** В состав команд арифметических операций микропроцессора входят команды байтовых операций суммирования, вычитания, десятичной коррекции, сравнения, инкремента и декремента, а также команды операций с двухбайтными операциями: суммирования, инкремента и декремента. При выполнении байтовых операций устанавливаются все флаги, при выполнении же команд операций инкремента и декремента регистровых пар флаги не устанавливаются, при выполнении команд суммирования с двухбайтными операндами устанавливается флаг переноса.

При выполнении байтовых операций суммирования и вычитания один из операндов должен быть помещен в аккумулятор, второй операнд может находиться в одном из регистров В,С,Д,Е,Н,Л, ячейке памяти или задаваться непосредственно. Результат операции всегда помещается в аккумулятор. Соответственно при



выполнении байтовых арифметических операций могут использоваться следующие режимы адресации:

1. Регистровая, например: ADD B (суммировать содержимое аккумулятора с содержимым регистра B);
2. Непосредственная, например: ADI 4Dh (суммировать содержимое аккумулятора с непосредственно заданным шестнадцатеричным операндом 4D).
3. Косвенная, например: ADD M (суммировать содержимое аккумулятора с содержимым ячейки памяти, адресуемой регистровой парой HL).

При выполнении операций с двухбайтными операндами один из них должен быть помещен в регистровую пару HL, а второй может находиться в одной из регистровых пар BC, DE или HL, результат помещается в регистровую пару HL, например: DAD B (суммировать содержимое регистровых пар HL и BC).

При выполнении арифметических операций над многобайтными данными вычисления выполняются побайтно, начиная с младшего байта. При этом используются команды операций сложения (вычитания) с учетом переноса в старший байт (заема из старшего байта), которые формируются в виде признака переноса C (Carry), например: ADC E (суммировать содержимое аккумулятора с содержимым регистра E, к результату прибавить значение флага переноса, который сформировался при выполнении предыдущей операции); SBI 25h (из содержимого аккумулятора вычесть непосредственно заданный шестнадцатеричный операнд 25h, из результата вычесть значение флага переноса, сформированного при выполнении предыдущей операции).

При выполнении операций инкремента (декремента) происходит увеличение (уменьшение) на единицу содержимого регистра (регистровой пары: например:

INR B (увеличение на 1 содержимого регистра B; устанавливаются флаги C, S, Z);

DCX H (уменьшение на единицу содержимого регистровой пары HL; флаги не устанавливаются).

При выполнении операции сравнения содержимое аккумулятора сравнивается с содержимым регистра, ячейки памяти или непосредственно заданным операндом; содержимое аккумулятора не изменяется, устанавливаются флаги S, Z), например: CMP D (сравнение содержимого аккумулятора с содержимым регистра D; если равно, устанавливается признак Z; если содержимое аккумулятора меньше содержимого регистра D, то устанавливается признак S).

При выполнении операций над данными, представленными в двоично-десятичном коде (BCD - binary coded decimal), после каждой операции необходимо выполнить десятичную коррекцию результата, которая заключается в добавлении к каждой тетраде числа 6 (0110), если содержимое тетрады превышает число 9, например:

Сложение чисел  $17 + 15 = 32$

Число 17 в BCD коде 00010111, число 15 – 00010101, число 32 - 00110010

$$\begin{array}{r}
 00010111 \\
 + \\
 00010101 \\
 \hline
 00101100 \\
 + \\
 00000110 \\
 \hline
 00110010
 \end{array}$$

**Задание:**

1. Загрузить программу 8080SDE.
2. Выполнить следующие последовательности операций (таблица 3.4) в пошаговом режиме; после выполнения каждой команды записать состояния всех регистров (таблица 3.5):

Таблица 3.4

Вариант 1	Вариант 2	Вариант 3	Вариант 4
<b>ORG 00h</b> <b>MVI C,00h</b> <b>MVI A, 26h</b> <b>LXI H, 7D2Bh</b> <b>MVI M, 4Fh</b> <b>ADD M</b> <b>DCR M</b> <b>SUB M</b> <b>MVI B, 7Fh</b> <b>ADD B</b> <b>INR C</b> <b>DCR B</b> <b>SUB C</b> <b>INX H</b> <b>DAD B</b> <b>ADI DAh</b> <b>SBB C</b> <b>MVI A,47h</b> <b>CPI A7h</b> <b>MVI D, 39h</b> <b>ADD D</b> <b>DAA</b> <b>SUI 28h</b> <b>DAA</b> <b>SUB A</b> <b>END</b>	<b>ORG 00h</b> <b>MVI C,00h</b> <b>MVI A, 45h</b> <b>LXI H, A5D2h</b> <b>MVI M, 32h</b> <b>ADD M</b> <b>DCR M</b> <b>SUB M</b> <b>MVI B, A3h</b> <b>ADD B</b> <b>INR C</b> <b>DCR B</b> <b>SUB C</b> <b>INX H</b> <b>DAD B</b> <b>ADI 35h</b> <b>SBB C</b> <b>MVI A,29h</b> <b>CPI 0Fh</b> <b>MVI D, 48h</b> <b>ADD D</b> <b>DAA</b> <b>SUI 38h</b> <b>DAA</b> <b>SUB A</b> <b>END</b>	<b>ORG 00h</b> <b>MVI C,00h</b> <b>MVI A, 37h</b> <b>LXI H, 4BA4h</b> <b>MVI M, 6Ah</b> <b>ADD M</b> <b>DCR M</b> <b>SUB M</b> <b>MVI B, B5h</b> <b>ADD B</b> <b>INR C</b> <b>DCR B</b> <b>SUB C</b> <b>INX H</b> <b>DAD B</b> <b>ADI 13h</b> <b>SBB C</b> <b>MVI A,38h</b> <b>CPI 5Fh</b> <b>MVI D, 54h</b> <b>ADD D</b> <b>DAA</b> <b>SUI 58h</b> <b>DAA</b> <b>SUB A</b> <b>END</b>	<b>ORG 00h</b> <b>MVI C,00h</b> <b>MVI A, A3h</b> <b>LXI H, 2ABDh</b> <b>MVI M, 2Eh</b> <b>ADD M</b> <b>DCR M</b> <b>SUB M</b> <b>MVI B, 47h</b> <b>ADD B</b> <b>INR C</b> <b>DCR B</b> <b>SUB C</b> <b>INX H</b> <b>DAD B</b> <b>ADI 9Fh</b> <b>SBB C</b> <b>MVI A,19h</b> <b>CPI 19h</b> <b>MVI D, 56h</b> <b>ADD D</b> <b>DAA</b> <b>SUI 47h</b> <b>DAA</b> <b>SUB A</b> <b>END</b>

Таблица 3.5

Адрес	Команда	Коды	Содержимое регистров								Комментарии
			A	B	C	D	E	H	L	F	

**Содержание отчета:**

1. Содержимое ячеек памяти после трансляции программы;
2. Таблица хода пошагового выполнения программы (таблица 3.5).

### Контрольные вопросы

1. Режимы адресации, используемые в командах арифметических операций.
2. Какие флаги формируются при выполнении арифметических операций?
3. Как можно обнулить содержимое регистра D?
4. Как осуществить суммирование содержимого двух ячеек памяти?
5. Как можно сравнить содержимое регистра и ячейки памяти?
6. Как можно сложить содержимое регистровых пар BC и DE?
7. Какой признак сформируется при выполнении операции SUB A?
8. Для чего используются операции суммирования с переносом?

## ЛАБОРАТОРНАЯ РАБОТА № 3 КОМАНДЫ ЛОГИЧЕСКИХ ОПЕРАЦИЙ МП i8080

**Цель работы:** Изучение команд логических операций

**Краткие теоретические сведения.** К командам логических операций относятся команды операций конъюнкции, дизъюнкции, суммирования по модулю 2, инверсии и сдвигов. При выполнении логических операций устанавливаются флаги нуля, переноса и паритета. В командах логических операций используются регистровый, косвенный, непосредственный и неявный способы адресации. При использовании регистровой, косвенной и непосредственной адресации один из операндов должен быть помещен в аккумулятор, при неявной адресации операндом является содержимое аккумулятора (инверсия и сдвиги). Результат операции всегда формируется в аккумуляторе.

Операции конъюнкции, дизъюнкции и суммирования по модулю 2 выполняются поразрядно, т.е. над одноименными разрядами операндов.

Примеры выполнения логических операций:

Конъюнкция содержимого аккумулятора с содержимым регистра B: ANA B;  
дизъюнкция содержимого аккумулятора с содержимым ячейки памяти, адресуемой регистровой парой HL: ORA M; суммирование по модулю 2 содержимого аккумулятора с непосредственно заданным операндом 4Bh: XRI 4Bh.

Пример:

Содержимое аккумулятора 01001101;

Содержимое регистра B 11011011

Операция XRA B

$$\begin{array}{r}
 01001101 \\
 \oplus \\
 11011011 \\
 \hline
 10010110
 \end{array}$$

Операция инверсии выполняется только над содержимым аккумулятора командой СМА, при этом инвертируются все разряды аккумулятора.

Операции сдвигов вправо и влево выполняются над содержимым аккумулятора. Существует два вида операций сдвига: сдвиг циклический и сдвиг через перенос. При циклической сдвиге вправо (влево) содержимое аккумулятора сдвигается по кольцу в сторону младших (старших) разрядов: соответственно младший (старший) разряд формирует флаг переноса С, при сдвиге вправо (влево) через перенос младший (старший) разряд аккумулятора формирует флаг переноса, а предыдущее значение флага переноса передается в старший (младший) разряд аккумулятора.

### Задание:

1. Загрузить программу 8080SDE.
2. Выполнить следующие последовательности операций (таблица 3.6) в пошаговом режиме; после выполнения каждой команды записать состояния всех регистров (таблица 3.7):

Таблица 3.6

Вариант 1	Вариант 2	Вариант 3	Вариант 4
<b>ORG 00h</b>	<b>ORG 00h</b>	<b>ORG 00h</b>	<b>ORG 00h</b>
<b>MVI A, 53h</b>	<b>MVI A, 37h</b>	<b>MVI A, A7h</b>	<b>MVI A, 53h</b>
<b>MVI B, 7Fh</b>	<b>MVI B, 3Dh</b>	<b>MVI B, 33h</b>	<b>MVI B, 7Fh</b>
<b>MVI C, DAh</b>	<b>MVI C, 25h</b>	<b>MVI C, ADh</b>	<b>MVI C, DAh</b>
<b>ANA C</b>	<b>ANA C</b>	<b>ANA C</b>	<b>ANA C</b>
<b>ORA B</b>	<b>ORA B</b>	<b>ORA B</b>	<b>ORA B</b>
<b>XRA C</b>	<b>XRA C</b>	<b>XRA C</b>	<b>XRA C</b>
<b>ANI 02h</b>	<b>ANI 01h</b>	<b>ANI 0Fh</b>	<b>ANI 02h</b>
<b>XRI 71h</b>	<b>XRI F2h</b>	<b>XRI 57h</b>	<b>XRI 71h</b>
<b>CMA</b>	<b>CMA</b>	<b>CMA</b>	<b>CMA</b>
<b>RAR</b>	<b>RAR</b>	<b>RAR</b>	<b>RAR</b>
<b>XRA A</b>	<b>XRA A</b>	<b>XRA A</b>	<b>XRA A</b>
<b>ORI 5Ah</b>	<b>ORI 05h</b>	<b>ORI 0fh</b>	<b>ORI 5Ah</b>
<b>RLC</b>	<b>RLC</b>	<b>RLC</b>	<b>RLC</b>
<b>STC</b>	<b>STC</b>	<b>STC</b>	<b>STC</b>
<b>CMC</b>	<b>CMC</b>	<b>CMC</b>	<b>CMC</b>
<b>XRA A</b>	<b>XRA A</b>	<b>XRA A</b>	<b>XRA A</b>
<b>LXI H, 5B3Dh</b>	<b>LXI H, 1F53h</b>	<b>LXI H, 13D0h</b>	<b>LXI H, 5B3Dh</b>
<b>MVI M, 04h</b>	<b>MVI M, 11h</b>	<b>MVI M, F0h</b>	<b>MVI M, 04h</b>
<b>MOV A,C</b>	<b>MOV A,C</b>	<b>MOV A,C</b>	<b>MOV A,C</b>
<b>ANA M</b>	<b>ANA M</b>	<b>ANA M</b>	<b>ANA M</b>
<b>DCX H</b>	<b>DCX H</b>	<b>DCX H</b>	<b>DCX H</b>
<b>MOV M,B</b>	<b>MOV M,B</b>	<b>MOV M,B</b>	<b>MOV M,B</b>
<b>ORA M</b>	<b>ORA M</b>	<b>ORA M</b>	<b>ORA M</b>
<b>RAL</b>	<b>RAL</b>	<b>RAL</b>	<b>RAL</b>
<b>ANI 00h</b>	<b>ANI 00h</b>	<b>ANI 00h</b>	<b>ANI 00h</b>
<b>ORI 07h</b>	<b>ORI 01</b>	<b>ORI 08h</b>	<b>ORI 80h</b>
<b>RRC</b>	<b>RRC</b>	<b>RRC</b>	<b>RRC</b>
<b>END</b>	<b>END</b>	<b>END</b>	<b>END</b>

Таблица 3.7

Адрес	Команда	Коды	Содержимое регистров								Комментарии
			A	B	C	D	E	H	L	F	

**Содержание отчета:**

1. Содержимое ячеек памяти после трансляции программы;
2. Таблица хода пошагового выполнения программы (таблица 3.7).

**Контрольные вопросы**

1. Способы адресации, используемые при выполнении команд логических операций; примеры написания команды с использованием различных способов адресации.
2. При выполнении каких операций формируется флаг переноса?
3. В чем отличие операций циклического сдвига и сдвига через перенос?
4. Как можно обнулить содержимое регистра, используя логические операции?
5. Как можно обнулить заданный разряд регистра?
6. Как можно установить в единицу заданный разряд регистра?
7. Как выполнить логическую операцию над содержимым двух ячеек памяти?
8. Как осуществить сдвиг содержимого регистровой пары?

## ЛАБОРАТОРНАЯ РАБОТА № 4

### КОМАНДЫ БЕЗУСЛОВНЫХ И УСЛОВНЫХ ПЕРЕХОДОВ МП KP580BM80A

**Цель работы:** изучение команд условных и безусловных переходов.

**Краткие теоретические сведения.** Операции переходов используются для организации ветвлений и циклов в программе. В состав системы команд входит команда безусловного перехода JMP addr и команды условных переходов J@@@ addr, где addr – двухбайтный адрес команды, к которой осуществляется переход (или символический адрес – метка), а символами @@@ обозначается признак, по которому выполняется условный переход. Безусловный переход по команде JMP addr аналогичен безусловным переходам, выполняемым в программах, написанных на языках высокого уровня, по команде “go to”. При выполнении этой команды управление передается команде, находящейся в памяти программ по адресу “addr”.

Условные переходы, выполняемые по командам “J@@@ addr”, аналогичные условным переходам, выполняемым в программах, написанных на языках высокого уровня, по командам “if <условие> go to”. Выполнение (или невыполнение) того или иного условия проверяется по состоянию соответствующего флага, формируемого в регистре признаков после выполнения предыдущей операции. Флаги,

по которым могут осуществляться переходы и соответствующие команды приведены в таблице 3.8.

Таблица 3.8.

Признак	Выполнение	Невыполнение
Перенос (Carry)	JC addr	JNC addr
Нуль (Zero)	JZ addr	JNZ addr
Знак (Plus, Minus)	JP addr	JM addr
Паритет (Parity: Even чет., Odd-нечет)	JPE addr	JPO addr

При обработке прерываний от внешних устройств (срабатывание датчиков, вмешательство оператора и т.п.) используются разновидности команд безусловного перехода – однобайтные так называемые команды “рестарта” RST N, в которых число N может принимать значения от 0 до 7. Код такой команды формируется внешним устройством и подается на шину данных процессора при одновременном формировании управляющего сигнала запроса прерывания “INTR”. По этой команде происходит запись в регистр команд числа Nx8, и соответственно передача управления по данному адресу. В ячейках памяти программ с данными адресами обычно записывается команда вызова программной процедуры обслуживания данного прерывания.

Примеры использования команд условных переходов для организации цикла:

1. Обнуление массива памяти объемом M байт с начальным адресом ADDR:

```

                LXI H, ADDR ;загрузка в регистровую пару HL начального адреса
XRA A          ;формирование в аккумуляторе нулевого байта
MVI B, M       ;организация в регистре B счетчика цикла
LOOP:          MOV M,A   ;начало цикла, обнуление первой ячейки массива
                INX H     ;инкремент адреса
                DCR B     ;декремент счетчика
                JNZ LOOP  ;если в регистре B не нуль, возврат в начало цикла

```

2. При разработке управляющих программ, программ обработки данных и т.п. часто требуется формирование временных интервалов заданной длительности. Такие интервалы формируются путем организации одного или нескольких вложенных циклов, в которых обычно выполняются либо незначащие операции, либо операции обращения к внешним устройствам (например, в программах ожидания события). Рассмотрим пример программы формирования временного интервала с использованием вложенных циклов. В качестве счетчиков циклов используем регистры D и E, в которые занесем числа N и M:

```

      MVI D, N
LOOP 2: MVI E, M
LOOP 1: DCR E
      NOP
      JNZ LOOP 1
      DCR D
      JNZ LOOP 2

```

Рассчитать длительность выполнения данного фрагмента можно зная время выполнения каждой команды: MVI R – 7 тактов; DCR R – 5 тактов; NOP – 4 такта; JNZ@@ - 10 тактов.

Первая команда выполняется однократно. Внутренний цикл выполняется М раз, внешний цикл – N раз. Таким образом, длительность выполнения данной программы можно определить:

$$T = 7 + N(7 + 10 + M(5 + 4 + 10)) = 7 + N(17 + 19M) \text{ тактов.}$$

При заданной величине Т, зная тактовую частоту процессора, легко можно подобрать целые числа N и М, чтобы обеспечить требуемую задержку. При необходимости можно добавлять дополнительные незначащие команды (например, NOP – пустая операция; пересылки, например, MOV A,A и т.п.).

### **Задание:**

1. Составить и выполнить программу заполнения массива памяти константой. (начальный адрес и размер массива задается преподавателем).
2. составить и выполнить программу формирования временного интервала заданной длительности.

### **Содержание отчета:**

- a) Блок-схема алгоритма программы;
- b) Содержимое ячеек памяти после трансляции и выполнения программы.

### **Контрольные вопросы**

1. Назначение команд условных и безусловных переходов.
2. По результатам каких операций можно выполнять условные переходы?
3. В чем отличие команд рестарта от условных и безусловных переходов?

## ЛАБОРАТОРНАЯ РАБОТА № 5

### КОМАНДЫ ВВОДА-ВЫВОДА И ОБРАЩЕНИЯ К ПОДПРОГРАММАМ. СТЕК.

**Цель работы:** изучение команд ввода-вывода, обращения к подпрограммам, работы со стеком.

**Краткие теоретические сведения.** Обращение процессора к внешним устройствам может осуществляться так же, как обращение к ячейкам памяти, с использованием прямой или косвенной адресации. При этом в качестве адреса используется двухбайтный адрес внешнего устройства. В то же время для обращения к внешним устройствам в состав команд микропроцессора введены специализированные команды ввода и вывода с использованием прямой адресации по однобайтному адресу внешнего устройства.

Команда ввода (INPUT) - IN Byte.

Команда вывода (OUTPUT) – OUT Byte. В этих командах Byte – однобайтный адрес внешнего устройства, который при выполнении команды выводится на шину адреса дважды, в старшем и младшем байтах адреса. При выполнении команды IN содержимое внешней шины данных записывается в аккумулятор, а при выполнении команды OUT содержимое аккумулятора выводится на внешнюю шину данных.

Команды обращения к стеку. При обработке прерываний, выполнении подпрограмм и других случаях используется стековая память, или стек. У некоторых микропроцессоров стек входит в состав блока регистров (аппаратный стек), у других формируется программным путем в области памяти данных (программный стек). У данного микропроцессора стек формируется программным путем при занесении в регистр-указатель стека (SP-Stack Pointer) начального адреса стековой области памяти (вершины стека) командой LXI SP ADDR. Стек организован по принципу “LIFO” – Last In – First Out – (последний вошел – первый вышел). В стек заносится содержимое регистровых пар командами:

PUSH B – запись в стек содержимого регистровой пары BC

PUSH D запись в стек содержимого регистровой пары DE

PUSH H - запись в стек содержимого регистровой пары HL

PUSH PSW запись в стек содержимого аккумулятора и регистра признаков PSW - (Program Status Word –слово состояния программы).

Данные записываются в стек по убывающим адресам, начиная с адреса ADDR –1. Запись содержимого регистровых пар может осуществляться в произвольном порядке, однако считывание из стека должно выполняться в порядке, противоположном порядку записи, и производится по команде POP:

POP PSW – возврат из стека содержимого аккумулятора и регистра флагов;

POP H - возврат из стека содержимого регистровой пары HL;

POP D возврат из стека содержимого регистровой пары DE;

POP B возврат из стека содержимого регистровой пары BC.



В состав команд микропроцессора входят команды безусловного и условного вызова подпрограмм. Безусловное обращение к подпрограмме осуществляется по команде CALL ADDR, где ADDR – адрес первой команды подпрограммы. В отличие от переходов при обращении к подпрограмме в стеке сохраняется адрес команды, следующей после команды CALL ADDR (адрес возврата). Безусловный возврат из подпрограммы осуществляется после команды RET, при этом из стека извлекается и помещается в счетчик команд адрес возврата и процессор выполняет команды, следующие после команды CALL.

Условное обращение к подпрограмме выполняется по команде C@@@, где @@ - условие, аналогичное используемым в командах условных переходов. Аналогично выход из подпрограммы может выполняться по условию @@, при этом используется команда условного возврата R@@. Подпрограмма может содержать несколько условных возвратов, однако последней командой в подпрограмме должна быть команда безусловного возврата RET.

Микропроцессор позволяет вложение подпрограмм, т.е. обращение из подпрограммы к другой, и нее к следующей и т.д., допуская до 8 вложений.

Область памяти, отводимая для размещения подпрограмм, не должна пересекаться с областями памяти данных и областью основной программы.

Пример использования подпрограммы: Прием массива данных объемом N байт из внешнего устройства с адресом IPORT и размещение в области памяти с начальным адресом ADDR.

	<b>LXI H, ADDR</b>	<b>; задание начального адреса памяти</b>
	<b>MVI B, N</b>	<b>; формирование счетчика байт</b>
<b>LOOP:</b>	<b>CALL READ</b>	<b>; вызов подпрограммы чтения данных</b>
	<b>INX H</b>	<b>; инкремент адреса</b>
	<b>DCR B</b>	<b>; декремент счетчика</b>
	<b>JNZ LOOP</b>	<b>; организация цикла</b>
.....		
.....		
<b>READ:</b>	<b>IN IPORT</b>	<b>; чтение байта данных</b>
	<b>MOV M,A</b>	<b>; запись байта в память</b>
	<b>RET</b>	<b>; возврат</b>

**Задание:** Составить и выполнить программу формирования сигнала с заданными длительностью и периодом следования, используя подпрограмму формирования временного интервала.

**Содержание отчета:**

1. Структурная схема алгоритма программы;
2. Текст программы;
3. Содержимое ячеек памяти после трансляции программы.

### Контрольные вопросы:

1. В чем отличие обращения к подпрограмме от условного или безусловного перехода?
2. Как можно обращаться к внешним устройствам?
3. Что поступает на шины данных и адреса при выполнении команд ввода-вывода?
4. Назначение стека.

## ЛАБОРАТОРНАЯ РАБОТА № 6 СОСТАВЛЕНИЕ И ОТЛАДКА ВЕТВЯЩИХСЯ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ КОМАНД ПЕРЕДАЧИ УПРАВЛЕНИЯ

**Цель работы** - изучить типовые алгоритмы составления ветвящихся программ и их реализацию в системе команд микропроцессора i8080 (KP580BM80A, ИК80A), научиться составлять простейшие ветвящиеся программы с использованием команд передачи управления и выполнять их с помощью 8080SDE.

**Теоретическая часть:** Типовые блок-схемы алгоритмов (БСА), широко встречающихся в программах обработки данных, приведены на рисунке 3.1.

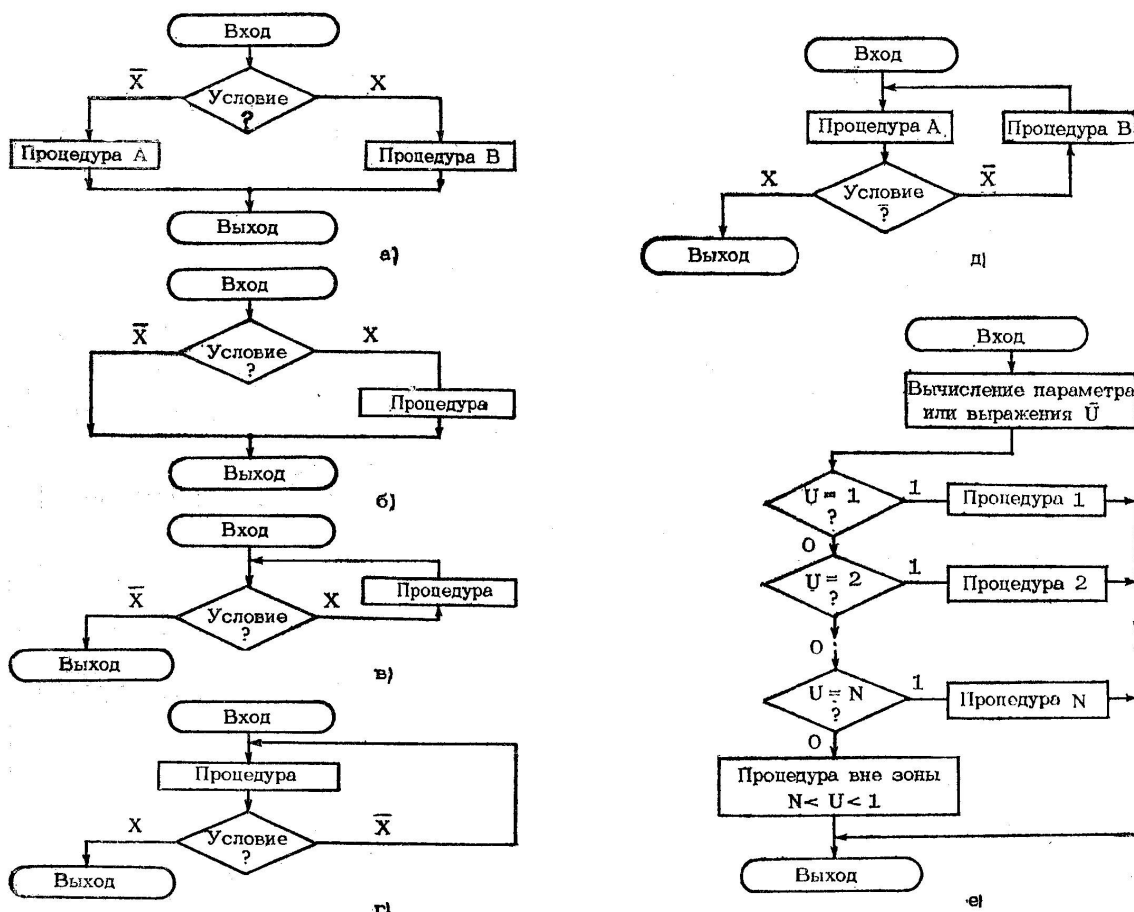


Рис. 3.1. Типовые блок-схемы алгоритмических структур: а - структура ЕСЛИ-ТО-ИНАЧЕ, б - структура ЕСЛИ-ТО, в - структура ДЕЛАЙ-ПОКА, г - структура ПО-

ВТОРАЯ-ДО-ТОГО-КАК, д - структура ПРОЦЕСС-ПОКА, е - структура ДЕЛАЙ-В-ЗАВИСИМОСТИ-ОТ.

Структура (а) применяется, когда нужно реализовать программный переход к одной из двух вычислительных процедур в зависимости от выполнения некоторого условия. Структура (б) используется, когда в зависимости от проверяемого условия нужно выполнить или не выполнить некоторую процедуру. Структуры (в) и (г) применяются для проверки окончания циклов, при этом процедура выполняется после (в) или до проверки условия (г). Структура (д) объединяет в себе две предыдущие структуры, а структура (е) осуществляет выбор действия при многозначных решениях и используется для замены цепочек структур типа (а). Для организации подобных структур программ предназначены команды передачи управления (см. раздел 1.3), которые нужно изучить перед выполнением работы.

В системе команд МП i8080 имеется обширный набор команд передачи управления (см. лабораторную работу № 4) Команда безусловной передачи управления JMP addr16 позволяет передать управление любой ячейке адресного пространства (загрузить ее содержимое в РС). Текущее содержимое РС при этом теряется. Эта команда соответствует оператору GOTO. Принципы структурного программирования рекомендуют обходиться без операторов такого типа.

Команды условной передачи управления передают управление только при выполнении некоторого условия, заданного в коде операции. Если оно не удовлетворяется, то перехода не происходит, а выполняется следующая по порядку команда. Проверяемым условием является текущее значение одного из флагов Z, C, S и P. Имеются команды перехода как по единичному, так и по нулевому значению флагов. Например, команда JZ передает управление, если флаг Z=1, а команда JNZ - если Z=0. Аналогичные парные команды имеются для флагов C (JC, JNC), S (JM, JP) и P (JPE, JPO). Всего получается 8 команд условных переходов. Особенной командой передачи управления является PCNL. Она передает управление по адресу, хранящемуся в регистровой паре HL.

В качестве примера организации условного перехода рассмотрим программу сбора данных и формирования в ОЗУ МП-системы одномерного массива однобайтовых данных. Одномерный массив - это набор элементов данных (записей) одинаковой длины, который размещается в области смежных ячеек памяти с начальным (базовым) адресом BASE. Число элементов в массиве называется его длиной. Положение каждого элемента в массиве задается его порядковым номером, называемым индексом, или смещением (отсюда - индексная адресация). Адрес элемента равен сумме базового адреса BASE и индекса IND, умноженного на длину элемента в байтах.

Формирование и обработка массивов осуществляется циклическими программами, состоящими из четырех частей: инициализация, обращение к текущему элементу, переход к следующему элементу и проверка условия окончания цикла. В программах, как правило, используются два регистра: регистр, где хранится адрес текущего элемента и называемый указателем (POINTER, PTR), и регистр-

счетчик, фиксирующий окончание цикла после обработки последнего элемента массива, если известна его длина. Пусть источник данных - порт с символическим адресом IPORT, BASE - начальный адрес массива данных. Регистр C будем использовать в качестве счетчика данных, а регистровую пару HL - в качестве указателя данных. Прием будем вести до появления в потоке данных знака “конец данных”, обозначенного как EOF. Блок-схема алгоритма (рис. 3.2) и текст программы приведены ниже. Поскольку при условных переходах анализируется содержимое регистра флагов, перед командами передачи управления должны находиться команды, устанавливающие анализируемые флаги.

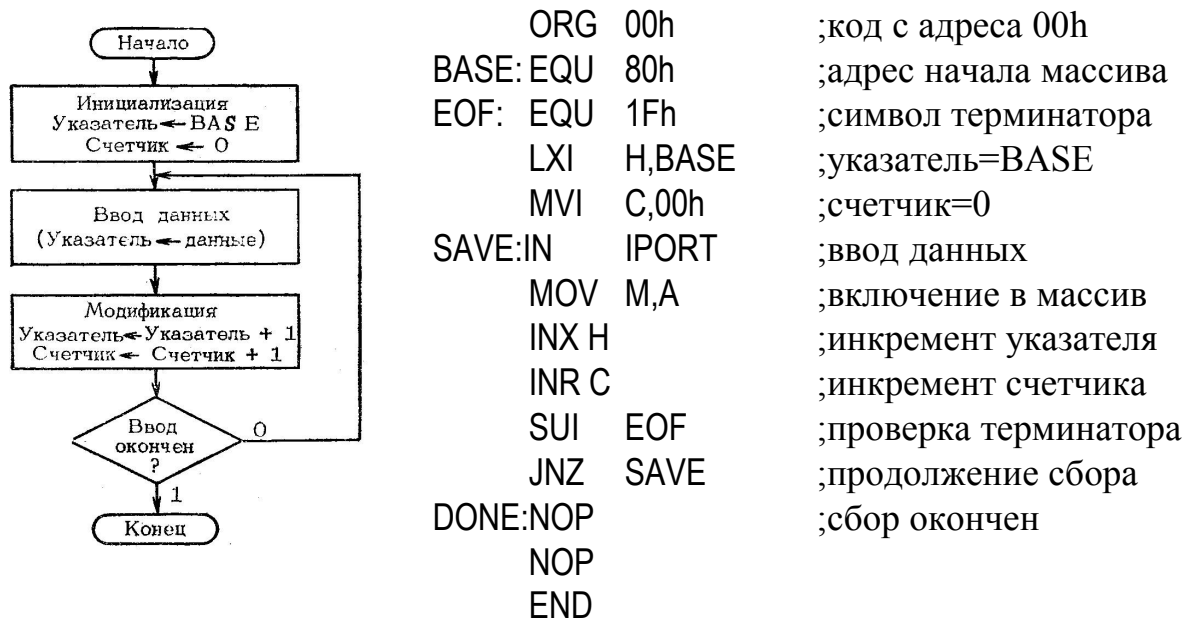
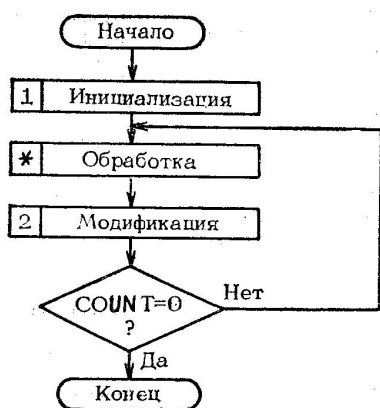


Рис. 3.2. БСА алгоритма

При обработке массивов часто требуется нахождение наибольшего и/или наименьшего значения его элементов. Простейший способ такой селекции таков: первый выбранный элемент объявляется наибольшим (наименьшим), последующие элементы сравниваются с этим числом, при обнаружении большего (меньшего) значения устанавливается новый максимум (минимум), равный этому новому значению. Процесс повторяется до достижения конца массива.

**Задание:** 1. Составьте следующие программы на языке Ассемблера в соответствии с приведенной универсальной БСА сбора и обработки данных (рис. 3.3).



Вариант 1: Программа нахождения суммы массива данных (без учета переполнения).

Вариант 2: Программа нахождения максимального значения в массиве целых однобайтных чисел без знака.

Вариант 3: Программа нахождения минимального значения в массиве целых однобайтных чисел без знака.

Рис.3.3. БСА сбора и обработки данных.

Вариант 4: Программа перемещения массива данных из одной области памяти в другую.

2. Загрузите программу 8080SDE, наберите текст программы, оттранслируйте его и выполните в пошаговом режиме. Массив данных из 10-15 элементов сформируйте вручную в некоторой области памяти с помощью HEX-вьюера.
3. Продемонстрируйте программу, ее работу и результат ее выполнения преподавателю.

### Форма отчета:

1. БСА программы;
2. Листинг программы с подробными комментариями;
3. Исходные данные и результаты выполнения программы в произвольной форме.

### Контрольные вопросы

1. Назовите типовые алгоритмические структуры. Для чего они применяются?
2. Что такое массив данных? Что такое длина и базовый адрес одномерного массива?
3. Чем определяется положение элемента в массиве и как определяется адрес элемента?
4. В чем состоит алгоритм нахождения максимального (минимального) элемента массива?

## ЛАБОРАТОРНАЯ РАБОТА № 7 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ТИПОВЫХ ФУНКЦИЙ УПРАВЛЕНИЯ И ВВОДА-ВЫВОДА В МП СИСТЕМЕ

**Цель работы** - изучить способы программной реализации типовых функций управления и ввода-вывода в микропроцессорной системе, научиться составлять простейшие программы с использованием команд ввода-вывода МП i8080 в МП-системе, эмулируемой с помощью 8080SDE.

**Теоретическая часть:** При проектировании микропроцессорных контроллеров возникает необходимость программирования таких типовых операций, как опрос состояния двоичного датчика, опрос состояния клавиатуры, ожидание события, формирование выходных управляющих сигналов типа “включить - выключить”, формирование временных задержек.

Алгоритмы опроса двоичного датчика и ожидания события сходны между собой. Схема аппаратной части такого датчика приведена на рис.3.4, а соответствующие БСА - на рис. 3.5.

Текст программы опроса датчика включает команды ввода в аккумулятор из порта с заданным адресом (в нашем примере 04), маскирование всех разрядов введенного байта, кроме  $D_5$ , переход к фрагменту LABEL A, если  $D_5=0$ :

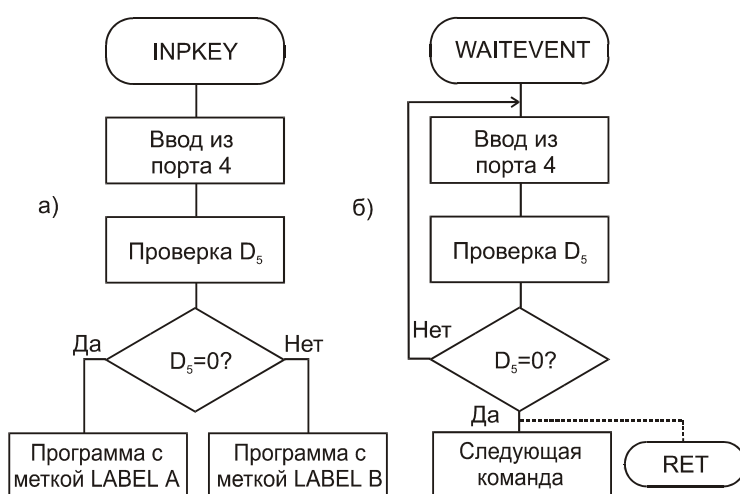
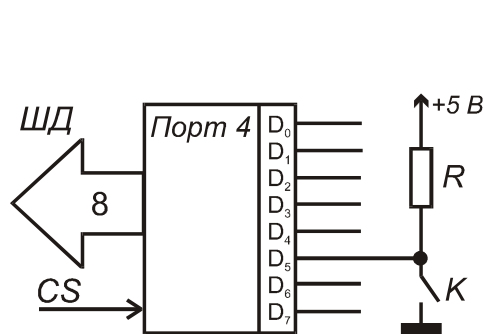


Рис. 3.4 Схема двоичного датчика.

Рис.3.5. Алгоритмы опроса датчика (а) и ожидания события (б)

```

INPKEY:  IN    04H      ;ввод
         ANI    20H      ;маскирование, маска 00100000
         JZ     LABEL A  ;переход, если 0
LABEL B:  ...           ;Начало фрагмента B
         ...
         ...
LABEL A:  ...           ;Начало фрагмента A

```

Аналогичным образом записывается программа ожидания события (БСА рис. 3.5 б).

Для формирования управляющих сигналов необходимо подключение некоторого порта вывода к исполнительному механизму. Формирование сигнала “включить” можно осуществить подачей на заданную линию выхода порта логической “1”, сигнала “выключить” - подачей логического “0”. Программа формирования сигнала, например, на выходе  $D_2$  порта с адресом 08H весьма проста:

**ON:**            **MVI   A,02H**            ; в аккумулятор - код 00000010  
                   **OUT   08H**            ; выдача в порт  
                   ...  
**OFF:**           **XRA   A**                    ; выключение: обнуляем аккумулятор  
                   **OUT   08H**            ; выдача в порт

Формирование временной задержки происходит обычно методом программных циклов и изучалось в лабораторной работе № 5. Задержки большой длительности можно получать методом вложенных циклов.

Комбинируя подпрограммы выдачи управляющих сигналов и временных задержек, можно получать на выходах портов последовательности импульсных сигналов с заданной длительностью и скважностью.

**Примечание:** Поскольку в эмуляторе 8080SDE время выполнения команд определяется многими факторами и сильно отличается от реальной МП системы в сторону увеличения, подбор задержек должен проводиться экспериментально.

**Применение таблиц в программах.** Таблицы являются удобной структурой данных и применяются для преобразования кодов, формирования текстовых сообщений (таблицы адресов), организации вычислений и т.д. Таблицы применяются также как средство структурирования программ (таблицы переходов). Таблица представляет собой обобщенный одномерный массив, элементы которого в общем случае могут иметь различную длину. В качестве примера применения таблиц рассмотрим табличное преобразование BCD-кода (двоично-десятичного кода) для семисегментных индикаторов (см. рис.2.8). Простой связи между этими кодами нет, поэтому удобно воспользоваться приведенной ниже таблицей. Преобразование заключается в обращении к таблице с использованием BCD кода как индекса.

Таблица преобразования BCD кодов в код семисегментных индикаторов

BCD-цифра				Код индикатора							
				D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	1	1	1	1	1	1
0	0	0	1	0	0	0	0	0	1	1	0
0	0	1	0	0	1	0	1	1	0	1	1
0	0	1	1	0	1	0	0	1	1	1	1
0	1	0	0	0	1	1	0	0	1	1	0
0	1	0	1	0	1	1	0	1	1	0	1
0	1	1	0	0	1	1	1	1	1	0	1
0	1	1	1	0	0	0	0	0	1	1	1
1	0	0	0	0	1	1	1	1	1	1	1
1	0	0	1	0	1	1	0	1	1	1	1

Программа преобразования BCD кода в код индикатора:

<b>CONV1:</b>	<b>LDA DIGIT</b>	<b>;цифра - в аккумулятор</b>
	<b>MOV E,A</b>	<b>;пересылка в регистр E</b>
	<b>MVI D,0</b>	<b>;сброс регистра D</b>
	<b>LXI H,BASE</b>	<b>;нач. адр. таблицы → в HL</b>
	<b>DAD D</b>	<b>;индексирование (HL)←(HL)+(DE)</b>
	<b>MOV A,M</b>	<b>;считывание кода индикатора</b>
	<b>STA CODE</b>	<b>;запоминание в ячейке CODE</b>

### Задания:

1. Ознакомиться с адресацией устройств ввода-вывода, эмулируемых программой 8080SDE (кнопочный регистр, светодиодные индикаторы, семисегментные индикаторы);

2. Составить следующие БСА и программы на языке Ассемблера:

Вариант 1: Программа, выполняющая по нажатию одной из кнопок включение “бегущего огня” светодиодов LD-индикатора и выключающая светодиоды при отжатии этой кнопки; Одна из кнопок должна соответствовать “бегущему огню” слева направо, другая - справа налево (по выбору).

Вариант 2: Программа, включающая каждый из светодиодов нажатием кнопки под ним; светодиоды должны работать в мигающем режиме; частота мигания должна быть минимальной для крайнего левого диода, каждый соседний диод должен мигать с частотой в 2 раза большей. При отжатии кнопки светодиод над ней должен погаснуть. При составлении программы используйте двоичный счетчик с задержкой и процедуру маскирования.

Вариант 3: Программа-счетчик, выводящая по нажатию одной из кнопок последовательно возрастающие через заданный промежуток времени числа от 0 до 9999 на семисегментный индикатор. Останов - по отжатию кнопки. Попробуйте подобрать время задержки так, чтобы значения возрастали с периодом 1 с - получится своего рода “секундомер”.

Вариант 4: Программа-индикатор кодовых комбинаций. Допустим, каждой кнопке из 8 кнопок LD-панели соответствует свой двоичный вес (справа налево 1-2-4-8-16-32-64-128), и эти кнопки образуют двоичный регистр ввода. Составить программу вывода на семисегментный индикатор числа от 0 до 255, соответствующего комбинации нажатых в данный момент кнопок. При отжатии/нажатии кнопок число должно соответствующим образом изменяться и соответствовать набранному кнопками двоичному коду.

Программы составить, оформляя процедуры ввода, вывода, задержек и т.д. в виде подпрограмм.



3. Отладить составленную программу и продемонстрировать ее работу преподавателю.

#### **Форма отчета:**

1. Блок-схема алгоритма программы;
2. Листинг программы с подробными комментариями;
3. Расчет временных задержек (если требуется).

#### **Контрольные вопросы**

1. Как можно осуществить выделение отдельных бит информационного слова при осуществлении операций ввода-вывода по отдельным линиям портов?
2. Для чего могут применяться таблицы в программах? Приведите примеры.

### **ЛАБОРАТОРНАЯ РАБОТА № 8 ВЫВОД ЗНАКОВОЙ И СИМВОЛЬНОЙ ИНФОРМАЦИИ НА МАТРИЧНЫЙ ИНДИКАТОР В МП СИСТЕМЕ**

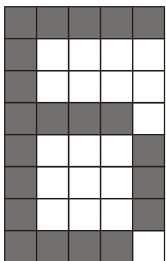
**Цель работы** - ознакомиться с некоторыми особенностями организации вывода знаковой и символьной информации и программной реализации функций динамического вывода такой информации, научиться составлять простейшие программы вывода информации в МП-системе, эмулируемой с помощью 8080SDE.

**Теоретическая часть:** Недостаток индикаторов семисегментного типа и других знаковых индикаторов является ограниченный набор выводимых символов. Другой тип индикаторов - матричные индикаторы - позволяет выводить значительно более разнообразную информацию, в том числе и в динамическом режиме. В системе, эмулируемой программой 8080SDE, есть полноцветный матричный индикатор 32×8 (32 столбца, 8 строк). Организация вывода информации на этот индикатор имеет много общего с выводом информации на растровый дисплей.

Излучающим элементам матричного индикатора в 8080SDE соответствует определенная область ОЗУ, называемая экранной областью. Начальный адрес этой области задается пользователем. Обозначим его BASE. Область делится на два сегмента: область активации пикселей (элементов изображения) и область цветовых атрибутов. Первый сегмент отвечает за включение соответствующих излучателей матрицы (светодиодов) и расположен по адресам от BASE до BASE+FFh. Заданный элемент включается размещением по соответствующему адресу кода FFh. Для выключения элемента заносится любой другой код. Номера строк (0-7) возрастают сверху вниз, номера столбцов (0-31, 00H-1Fh) - справа налево. Адрес элемента  $A_{ij}$ , расположенного в  $i$ -й строке и  $j$ -м столбце, определяется как  $A_{ij} = \text{BASE} + i * 2 + j$ . Область цветовых атрибутов занимает адреса от BASE+100h до BASE+3FFh и разделена на 3 сегмента: от BASE+100h до BASE+1FFh - яркость красного цвета, от BASE+200h до BASE+2FFh - яркость зеленого цвета, от

BASE+300h до BASE+3FFh - яркость синего цвета. Запись кода 00 соответствует нулевой, запись FFh - максимальной яркости. Всего, таким образом, можно получить  $FF \times FF \times FF = 255^3 = 16\,581\,375$  цветовых оттенков. Организация вывода информации заключается в записи требуемых значений по соответствующим адресам экранной области, помещении в регистровую пару HL значения BASE и подачи команды OUT 190<sub>10</sub> (OUT BEh), выполняющей роль строба записи.

**Выдача алфавитно-цифровой информации. Знакогенераторы.** Обычно алфавитно-цифровая (символьная) информация хранится в ОЗУ МП-системы в виде алфавитно-цифровых кодов. Существует несколько систем кодировки символов. Часто применяется система кодировки ASCII (аббревиатура от American Standard Code for Information Interchange). В этой системе символы кодируют байтами от 00h до FFh (всего 256 символов). Таблица кодов ASCII для MS-DOS представлена ниже (приведена так называемая альтернативная кодировка символов кириллицы, наиболее часто используемая и ставшая стандартом де-факто). Вывод на индикатор матричного или растрового типа требует перекодировки этих кодов в коды индикации, содержащие информацию об активных элементах раstra. Символы при этом кодируются в виде матриц 5×8, 6×8, 8×12 и т.д. Для примера ниже приведен символ Б, представленный в матрице 5×8. При программной перекодировке коду индицируемого символа должен быть поставлен в соответствие определенный код, описывающий вид такой матрицы для данного символа. Для нашего случая матрица может быть описана 5-ю байтами, в которых 1 соответствует активным (отображаемым) элементам, т.е. темным клеточкам на рисунке, а 0 - неактивным элементам.



Перекодировку удобнее всего проводить табличным способом. Для этого байты индикации (или наборы таких байтов), описывающие матрицы символов, располагаются в смежных ячейках памяти с возрастающими адресами. Такие таблицы называют таблицами знакогенераторов. Для описания матрицы 5×8 потребуется 5 байт памяти. В 8080SDE для кодирования каждого элемента матрицы требуется отдельный байт (а не бит), соответственно возрастает и объем таблицы знакогенератора. Для случая, когда каждый символ описывается несколькими байтами индикации (в нашем случае 5), требуется еще одна таблица перекодировки, ставящая в соответствие коду каждого символа начальный адрес расположения группы байтов индикации этого символа.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		☺	☹	♥	♦	♣	♠	•	◼	○	◼	♂	♀	♪	♫	☼
1	▶	◀	↕	!!	¥	§	_	↑	↓	→	←	+	⌈	⌋	▲	▼
2		!	"	#	\$	%	&	'	(	)	*	+	,	=	>	/
3	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
4	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
5	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
6																
7																
8	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
9	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
A	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т
B	г	д	е	ж	з	и	й	к	л	м	н	о	п	р	с	т
C	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т
D	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т
E	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т
F	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т

*Таблица ASCII-кодов*

Пример: код символа Г - 83h.

При выводе на экран или на матричный индикатор динамических (движущихся) изображений необходимо каждый раз последовательно “перерисовывать” либо весь экран, либо (для ускорения) только область, где изменилось изображение.

В качестве примера приведем программу, непрерывно выводящую на матричный дисплей движущийся справа налево вертикальный столбец красного цвета:

	<b>ORG 0000h</b>	<b>;адрес начала программы</b>
<b>BASE:EQU</b>	<b>100h</b>	<b>;адрес начала экранной области</b>
<b>START:</b>	<b>LXI D,0000h</b>	<b>;обнуляем счетчик столбцов</b>
<b>LOOP:</b>	<b>MVI C,FFh</b>	<b>;зажигаем</b>
	<b>CALL OUTCOL</b>	<b>;столбец</b>
	<b>MVI C,00h</b>	<b>;гасим</b>
	<b>CALL OUTCOL</b>	<b>;столбец</b>
	<b>INX D</b>	<b>;следующий столбец</b>
	<b>MOV A,E</b>	<b>;последний столбец</b>
	<b>CPI 20h</b>	<b>;уже пройден?</b>
	<b>JNZ LOOP</b>	<b>;нет, продолжаем</b>
	<b>JMP START</b>	<b>;да, все сначала</b>
<b>OUTCOL:</b>	<b>LXI H,BASE+100h</b>	<b>;адрес области атрибутов красного</b>

	DAD D	;определяем адрес столбца
	MVI B,08h	;устанавливаем счетчик строк
LOOP1:	MOV A,C	;выводимый байт
	MOV M,A	;помещаем по адресу элемента
	MVI A,20h	;наращиваем
	ADD L	;адрес
	MOV L,A	;строки
	DCR B	;декремент счетчика строк
	JNZ LOOP1	;если не последняя строка - дальше
	LXI H,BASE	;установка адреса экранной области
	OUT EBh	;строб вывода
	RET	;возврат из подпрограммы OUTCOL
	ORG 100h	;инициализация экранной области
	DS 256,FFh	;все элементы активны
	DS 256,00h	;гасим красный
	DS 256,00h	;гасим зеленый
	DS 256,00h	;гасим синий
	END	

#### **Задания:**

1. Ознакомиться с организацией вывода информации на матричный дисплей системы 8080SDE.
2. Составить следующие БСА и программы на языке Ассемблера:  
 Вариант 1: Программа, выводящая движущуюся горизонтальную линию произвольной длины, меняющую направление движения на противоположное по достижении границы дисплея.  
 Вариант 2: Программа, выводящая движущуюся вертикальную линию произвольной длины, меняющую направление движения на противоположное по достижении границы дисплея.  
 Вариант 3: Программа, выводящая движущийся горизонтально кубик 4×4, меняющий направление движения по достижении границы дисплея.  
 Вариант 4\*: Программа, выводящая движущийся справа налево буквенно-цифровой символ (5×8) или группу символов (бегущая строка). (\* *Задача повышенной сложности*)
3. Отладить составленную программу и продемонстрировать ее работу преподавателю.

#### **Форма отчета:**

1. Блок-схема алгоритма программы;
2. Листинг программы с подробными комментариями.

#### **Контрольные вопросы**

1. Как производится выдача символьной информации на растровые устройства индикации?
2. Что такое таблица знакогенератора и какая информация в ней содержится?
3. Как обычно кодируются символы в памяти МП-систем? Что такое коды ASCII? Сколько символов можно закодировать по этому стандарту?
4. Какой объем памяти займет описание одного символа в таблице знакогенератора, если символ представляется матрицей  $7 \times 8$  и каждый элемент матрицы кодируется одним байтом?
5. Опишите последовательность действий при выводе на растровое устройство индикации изменяющейся информации (движущийся объект).

Авторы выражают искреннюю благодарность Марку Балке (Marc Balke, Огайо, США) - автору программы 8080SDE, за предоставленную возможность использования этой программы.

## ЛИТЕРАТУРА

1. Токхайм Р. Микропроцессоры. Курс и упражнения / Пер. с англ., под ред. В.Н. Грасевича. - М.: Энергоатомиздат, 1988. - 336 с.
2. Григорьев В.Л. Программное обеспечение микропроцессорных систем. - М.: Энергоатомиздат, 1983. - 208 с.
3. Каган Б.М., Сташин В.В. Основы проектирования микропроцессорных устройств автоматики. - М.: Энергоатомиздат, 1987. - 304 с.
4. Шпота С.Д., Ломако Н.А. Изучение архитектуры и структуры однокристалльного микропроцессора (на базе микросхемы КР580ИК80): Лабораторный практикум по курсу "Цифровые устройства и устройства цифровой техники". - Мн.: БГУИР, 1998. - 66 с.
5. Лапшин С.М. Проектирование микропроцессорных систем передачи и обработки информации. Методическое пособие по курсовому и дипломному проектированию по курсам "Устройства цифровой техники" и "Микропроцессоры в системах телекоммуникаций" для студентов специальности "Телекоммуникационные системы". - Мн.: БГУИР, 1997. - 51 с.

## СОДЕРЖАНИЕ

	Стр.
<b>1. СИСТЕМА КОМАНД МИКРОПРОЦЕССОРА i8080 (КР580ВМ80А, КР580ИК80А) .....</b>	<b>3</b>
<b>2. ЭМУЛЯТОР МИКРОПРОЦЕССОРНОЙ СИСТЕМЫ 8080SDE.....</b>	<b>13</b>
<b>3. ЛАБОРАТОРНЫЙ ПРАКТИКУМ.....</b>	<b>23</b>
<b>ЛАБОРАТОРНАЯ РАБОТА № 1 .....</b>	<b>23</b>
<b>ЛАБОРАТОРНАЯ РАБОТА № 2.....</b>	<b>25</b>
<b>ЛАБОРАТОРНАЯ РАБОТА № 3 .....</b>	<b>28</b>
<b>ЛАБОРАТОРНАЯ РАБОТА № 4 .....</b>	<b>30</b>
<b>ЛАБОРАТОРНАЯ РАБОТА № 5 .....</b>	<b>33</b>
<b>ЛАБОРАТОРНАЯ РАБОТА № 6 .....</b>	<b>35</b>
<b>ЛАБОРАТОРНАЯ РАБОТА № 7 .....</b>	<b>38</b>
<b>ЛАБОРАТОРНАЯ РАБОТА № 8 .....</b>	<b>42</b>
<b>ЛИТЕРАТУРА .....</b>	<b>46</b>
<b>СОДЕРЖАНИЕ .....</b>	<b>47</b>