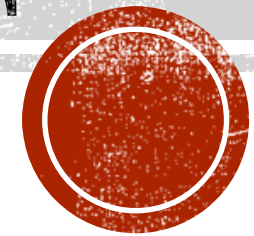
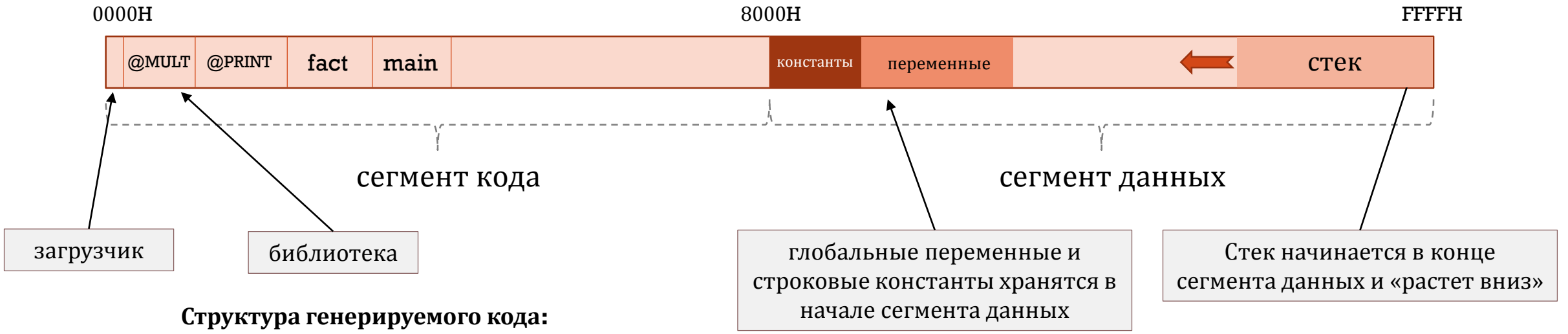
ЯЗЫКИ ПРОГРАММИРОВАНИЯ И МЕТОДЫ ТРАНСЛЯЦИИ



Лекции *9-10*

20 апреля 2018 г.

КРАТКОЕ ОПИСАНИЕ МАШИНЫ 8080



Структура генерируемого кода:

ORG 8000H	Перемещаем счетчик ассемблера в сегмент данных
`var0: DB 0` `var1: DB 0`	Резервируем ячейки памяти под все глобальные переменные. Ставим метки, чтобы можно было обращаться к этим ячейкам памяти
ORG 0	Возвращаем счетчик ассемблера на начало сегмента кода

`LXI H, 0`
`SPHL`
`CALL main`
`END`
`@MULT: …`
`@PRINT: …`
`fact: …`
`main: …`
	Генерируем остальной код

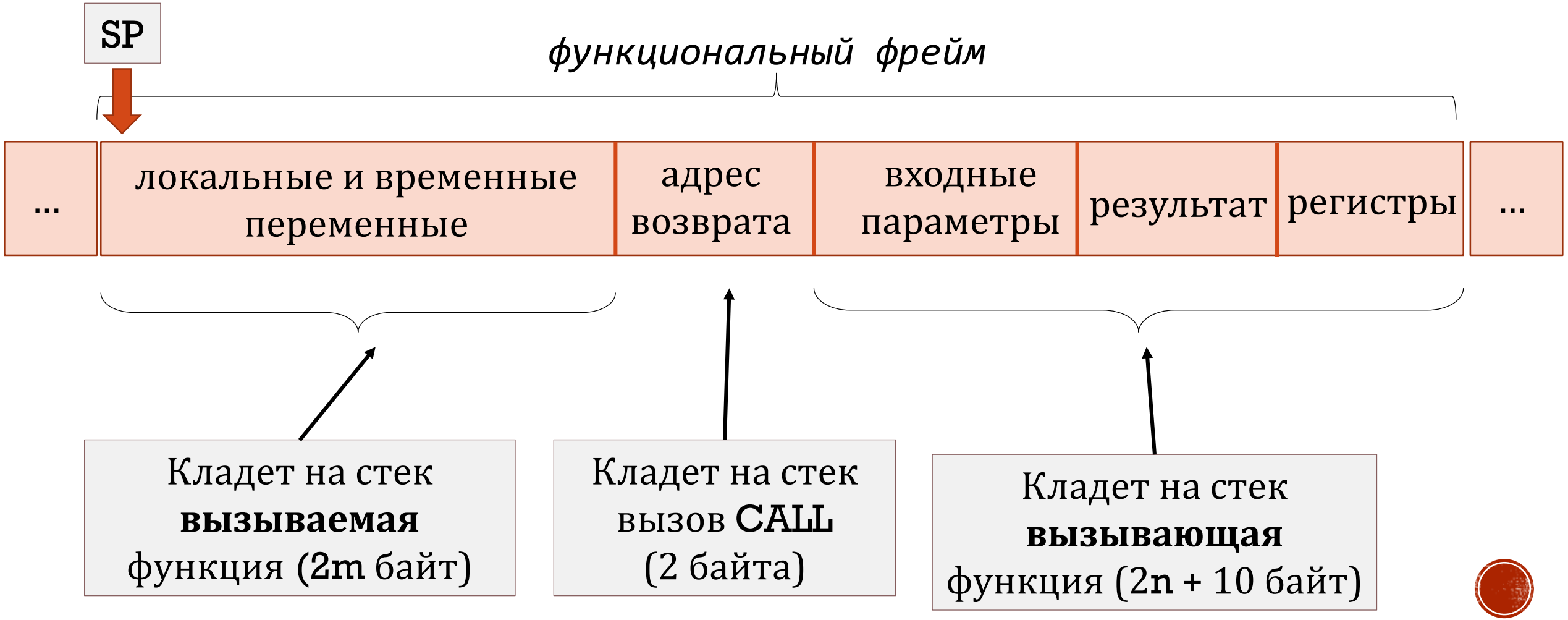
Если после трансляции в ТС не оказалось функции **main** – ошибка!

```
SYMBOL TABLE
-------------------------------------------------------
code      name      kind      type      len       default   scope
0         a         var       int       None      0         -1
1         b         var       int       None      None      -1
2         fact      func      int       1         None      -1
3         n         None      None      None      None      -1
4         n         var       int       None      None      2
5         main      func      int       0         None      -1
6         x         None      None      None      None      -1
7         x         var       int       None      None      5
8         a         var       int       None      None      5
```

СТЕК

Размер
ffsize = 2m + 12 + 2n

Стек состоит из функциональных фреймов:



СТЕК

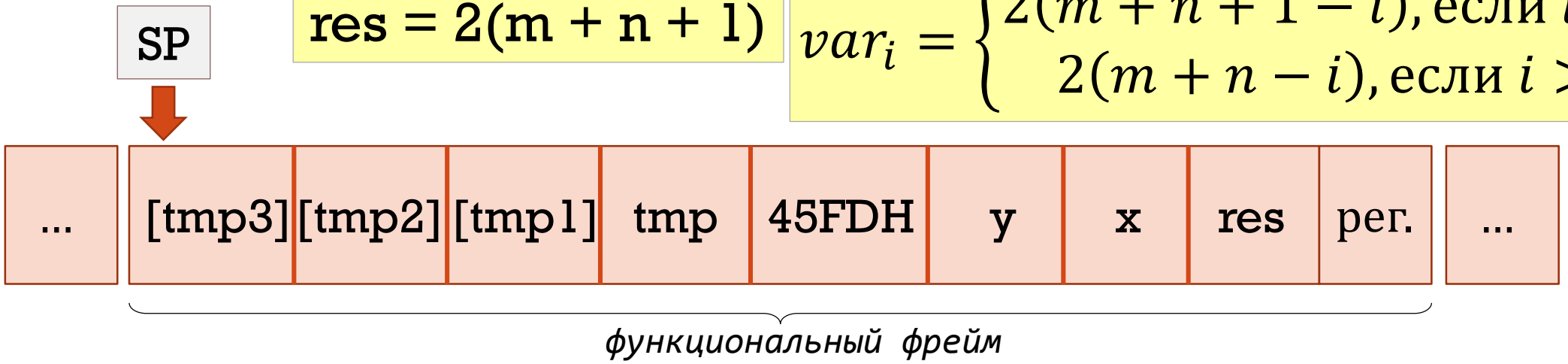
n = len

vars = sum("var в scope")

m = vars - n

res = 2(m + n + 1)

$$var_i = \begin{cases} 2(m+n+1-i), \text{если } i \le n, \\ 2(m+n-i), \text{если } i > n \end{cases}$$




```
int func(int x, int y){
   int tmp = 0;
   tmp = (x + y) * 2;
   if (tmp < y) tmp = y;
   return tmp;
}
```

code	name	kind	type	len	default	scope	offset
0	func	func	int	2	None	-1	-1
1	x	None	None	None	None	-1	-1
2	x	var	int	None	None	0	12
3	y	None	None	None	None	-1	-1
4	y	var	int	None	None	0	10
5	tmp	None	None	None	None	-1	-1
6	tmp	var	int	None	0	0	6
7	[tmp1]	var	int	None	None	0	4
8	[tmp2]	var	int	None	None	0	2
9	[tmp3]	var	int	None	None	0	0
10	main	func	int	0	None	-1	-1

ОБЩАЯ СХЕМА ГЕНЕРАЦИИ КОДА

Для генерации кода атома (ATOM, op1, op2, op3) необходимо:

1) Получить первый операнд – op1
2) Сохранить первый операнд в свободном регистре
3) Получить второй операнд – op2
4) Выполнить операцию, задаваемую типом атома – ATOM
5) Сохранить результат в месте, указываемым op3

АДРЕСАЦИЯ

Есть одиннадцать вариантов того, чем может быть opX:

1. Числовая или символьная константа (num)
2. Строковая константа (str)
3. Глобальная переменная (id)
4. Локальная переменная (id)
5. Глобальный массив с константным индексом mas[num]
6. Глобальный массив с глобальным индексом mas[id]
7. Глобальный массив с локальным индексом mas[id]
8. Локальный массив с константным индексом mas[num]
9. Локальный массив с глобальным индексом mas[id]
10. Локальный массив с локальным индексом mas[id]
11. Метка

Используется только для **получения** операнда, не для записи

ОПЕРАНД АТОМА

```
class AOpType(Enum):
    mem=1; num=2; str=3; lbl=4; arr=5

class AOp:
    def __init__(self, type = None, value = None, index = None):
        self.type = type
        self.value = value
        self.index = index
```

ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ

Вспомогательные функции для адресации:

- void **loadOp**(AOp op) – в зависимости от типа операнда op генерирует код, записывающий в регистр A значение операнда

- void **saveOp**(AOp op) – в зависимости от типа операнда op генерирует код, сохраняющий значение регистра A в месте, на которое указывает операнд

Вспомогательные функции для работы с регистрами:

- void **saveRegs**() – записывает все регистры в стек
- void **loadRegs**() – достает все регистры из стека

АДРЕСАЦИЯ

Десять вариантов того, чем может быть opX:

1. Числовая или символьная константа (num)
2. Строковая константа (str)
3. Глобальная переменная (id)
4. Локальная переменная (id)
5. Глобальный массив с константным индексом mas[num]
6. Глобальный массив с глобальным индексом mas[id]
7. Глобальный массив с локальным индексом mas[id]
8. Локальный массив с константным индексом mas[num]
9. Локальный массив с глобальным индексом mas[id]
10. Локальный массив с локальным индексом mas[id]

КРАТКОЕ ОПИСАНИЕ МАШИНЫ 8080

Перемещения

MOV regm,regm	regm = regm
STAX rp LDAX rp	(rp) = A ;только B,D A = (rp) ;только B,D
LXI rp,data16	rp = data16 ;B,D,H,PSW,SP
MVI regm,data	regm = data
STA addr LDA addr	(addr) = A A = (addr)
SHLD addr LHLD addr	(addr)=L (addr+1)=H L=(addr) H=(addr+1)

Арифметико-логические операции

ADD regm ADC regm SUB regm SBB regm ANA regm XRA regm ORA regm CMP regm	A = A + regm A = A + regm + carry A = A - regm A = A - regm - carry A = A AND regm A = A XOR regm A = A OR regm [A – regm]
ADI data ACI data SUI data SBI data ANI data XRI data ORI data CPI data	A = A + data A = A + data + carry A = A - data A = A - data - carry A = A AND data A = A XOR data A = A OR data [A – data]

Прыжки

PCHL	pc = H:L
JMP addr	безусловный переход на addr
JC addr JNC…, JZ…, JNZ…, JP…, JM…, JPE…, JPO…	условный переход: carry, no carry, zero, no zero, plus, minus, even, odd

Стек

PUSH rp	(sp-1)=rp1 (sp-2)=rp2 sp=sp-2
POP rp	rp1=(sp-1) rp2=(sp-2) sp=sp+2
XCHG	h <-> d l <-> e
XTHL	l <-> (sp) h <-> (sp)+1
SPHL	sp = H:L

Вызов и возврат из функции

CALL addr	безусловный **вызов addr**
CC…, CNC…, CZ…, CNZ…, CP…, CM…, CPE…, CPO…	условный переход: carry, no carry, zero, no zero, plus, minus, even, odd
RET	безусловный **возврат**
RC, RNC, RZ, RNZ, RP, RM, RPE, RPO	условный возврат: carry, no carry, zero, no zero, plus, minus, even, odd

Унарные и прочие операции

INR regm DCR regm	regm = regm + 1 regm = regm – 1
CMA	A = ~A
INX rp DCX rp	rp = rp + 1 rp = rp – 1
DAD rp	H:L = H:L + sp
HALT	Остановка работы

Ввод/вывод (8 бит)

IN 0	ввод в A (8 бит)
OUT 1	вывод из A (8 бит)

Сдвиги

RLC RRC	нециклические сдвиги
RAL RAR	циклические сдвиги

Мета-инструкции

DB list	Резервирование памяти под каждый байт из list
DW list	Резервирование памяти под каждое слово из list
DS exp	Резерв. exp байт памяти
ORG data16	Перемещение указателя ассемблирования
END	Конец программы

Условные обозначения

regm	Регистры **A**,**B**,**C**,**D**,**E**,**H**,**L** или буква **M** (значение берется из я.п., адрес которой хранится в **h**,**l**
rp (rp)	Регистровая пара: **B** (=**B**:**C**) **D** (=**D**:**E**), **H** (=**H**:**L**), **PSW** (=флаги:**A**), **SP** Значение ячейки памяти по адресу, записанному в паре **rp**
data	8-битная константа (0-255)
data16	16-битная константа (0-65 536)
addr (addr)	Адрес памяти (16-битная константа или метка) Значение ячейки памяти по адресу **addr**
sp	Указатель на вершину стека
pc	Счетчик команд
carry	Бит переноса
H:L	Регистровая пара

ОБЩАЯ СХЕМА ГЕНЕРАЦИИ КОДА ФУНКЦИИ

Перед началом генерации кода для тела функции:

1. Ставим метку на текущую ячейку с именем функции
2. Проходим по таблице символов и для каждой локальной и временной переменной выводим соответствующее количество PUSH (для обычных переменных – один, для массивов – n, где n – размер массива)

После генерации кода для тела функции:

Ничего не делаем, так как нужный код будет сгенерирован благодаря «запасной» инструкции RET, вставленной правилом №2 транслирующей грамматики основных конструкций языка MiniC.

Все атомы для функций кладем в отдельные списки.

ГЕНЕРАЦИЯ КОДА АТОМОВ

1. (ADD, x, y, z)
2. (MUL, x, y, z)
3. (MOV, x, ,, z)
4. (LBL,,, L1)
5. (EQ, x, y, L1)
6. (JMP,,, L1)
7. (IN,,, x)
8. (OUT,,, x)
9. (RET,,, x)
10. (PARAM,,, x)
11. (CALL, x,, y)

УПРАЖНЕНИЯ

Упражнение 1

Напишите таблицу символов, список атомов

```
int a;
int main(){
  int b;
  in a; in b;
  int result[2];
  result[0] = func(a, b);
  out result[0];
  return 0;
}
```

```
int func(int x, int y){
  int tmp = 0;
  tmp = (x + y) * 2;
  if (tmp < y) tmp = y;
  return tmp;
}
```

УПРАЖНЕНИЯ

```
0     (ADD, 2, 4, 7)
0     (MUL, 7, '2', 8)
0     (MOV, 8, , 6)
0     (MOV, '1', , 9)
0     (LT, 6, 4, L2)
0     (MOV, '0', , 9)
0     (LBL, , , L2)
0     (EQ, 9, '0', L0)
0     (MOV, 4, , 6)
0     (JMP, , , L1)
0     (LBL, , , L0)
0     (LBL, , , L1)
0     (RET, , , 6)
0     (RET, , , '0')
11    (IN, , , 10)
11    (IN, , , 13)
11    (PARAM, , , 13)
11    (PARAM, , , 10)
11    (CALL, 0, , 16)
11    (MOV, 16, , 15['0'])
11    (OUT, , , 15['0'])
11    (RET, , , '0')
11    (RET, , , '0')
```

code	name	kind	type	len	default	scope
0	func	func	int	2	None	-1
1	x	None	None	None	None	-1
2	x	var	int	None	None	0
3	y	None	None	None	None	-1
4	y	var	int	None	None	0
5	tmp	None	None	None	None	-1
6	tmp	var	int	None	0	0
7	[tmp1]	var	int	None	None	0
8	[tmp2]	var	int	None	None	0
9	[tmp3]	var	int	None	None	0
10	a	var	int	None	None	-1
11	main	func	int	0	None	-1
12	b	None	None	None	None	-1
13	b	var	int	None	None	11
14	result	None	None	None	None	-1
15	result	array	int	2	None	11
16	[tmp4]	var	int	None	None	11

КРАТКОЕ ОПИСАНИЕ МАШИНЫ 8080

Перемещения

MOV regm,regm	regm = regm
STAX rp LDAX rp	(rp) = A ;только B,D A = (rp) ;только B,D
LXI rp,data16	rp = data16 ;B,D,H,PSW,SP
MVI regm,data	regm = data
STA addr LDA addr	(addr) = A A = (addr)
SHLD addr LHLD addr	(addr)=L (addr+1)=H L=(addr) H=(addr+1)

Арифметико-логические операции

ADD regm ADC regm SUB regm SBB regm ANA regm XRA regm ORA regm CMP regm	A = A + regm A = A + regm + carry A = A - regm A = A - regm - carry A = A AND regm A = A XOR regm A = A OR regm [A – regm]
ADI data ACI data SUI data SBI data ANI data XRI data ORI data CPI data	A = A + data A = A + data + carry A = A - data A = A - data - carry A = A AND data A = A XOR data A = A OR data [A – data]

Прыжки

PCHL	pc = H:L
JMP addr	безусловный переход на addr
JC addr JNC…, JZ…, JNZ…, JP…, JM…, JPE…, JPO…	условный переход: carry, no carry, zero, no zero, plus, minus, even, odd

Стек

PUSH rp	(sp-1)=rp1 (sp-2)=rp2 sp=sp-2
POP rp	rp1=(sp-1) rp2=(sp-2) sp=sp+2
XCHG	h <-> d l <-> e
XTHL	l <-> (sp) h <-> (sp)+1
SPHL	sp = H:L

Вызов и возврат из функции

CALL addr	безусловный **вызов addr**
CC…, CNC…, CZ…, CNZ…, CP…, CM…, CPE…, CPO…	условный переход: carry, no carry, zero, no zero, plus, minus, even, odd
RET	безусловный **возврат**
RC, RNC, RZ, RNZ, RP, RM, RPE, RPO	условный возврат: carry, no carry, zero, no zero, plus, minus, even, odd

Унарные и прочие операции

INR regm DCR regm	regm = regm + 1 regm = regm – 1
CMA	A = ~A
INX rp DCX rp	rp = rp + 1 rp = rp – 1
DAD rp	H:L = H:L + sp
HALT	Остановка работы

Ввод/вывод (8 бит)

IN 0	ввод в A (8 бит)
OUT 1	вывод из A (8 бит)

Сдвиги

RLC RRC	нециклические сдвиги
RAL RAR	циклические сдвиги

Мета-инструкции

DB list	Резервирование памяти под каждый байт из list
DW list	Резервирование памяти под каждое слово из list
DS exp	Резерв. exp байт памяти
ORG data16	Перемещение указателя ассемблирования
END	Конец программы

Условные обозначения

regm	Регистры **A**,**B**,**C**,**D**,**E**,**H**,**L** или буква **M** (значение берется из я.п., адрес которой хранится в **h**,**l**
rp (rp)	Регистровая пара: **B** (=**B**:**C**) **D** (=**D**:**E**), **H** (=**H**:**L**), **PSW** (=флаги:**A**), **SP** Значение ячейки памяти по адресу, записанному в паре **rp**
data	8-битная константа (0-255)
data16	16-битная константа (0-65 536)
addr (addr)	Адрес памяти (16-битная константа или метка) Значение ячейки памяти по адресу **addr**
sp	Указатель на вершину стека
pc	Счетчик команд
carry	Бит переноса
H:L	Регистровая пара