Содержание

1. Система команд

1.1. Форматы

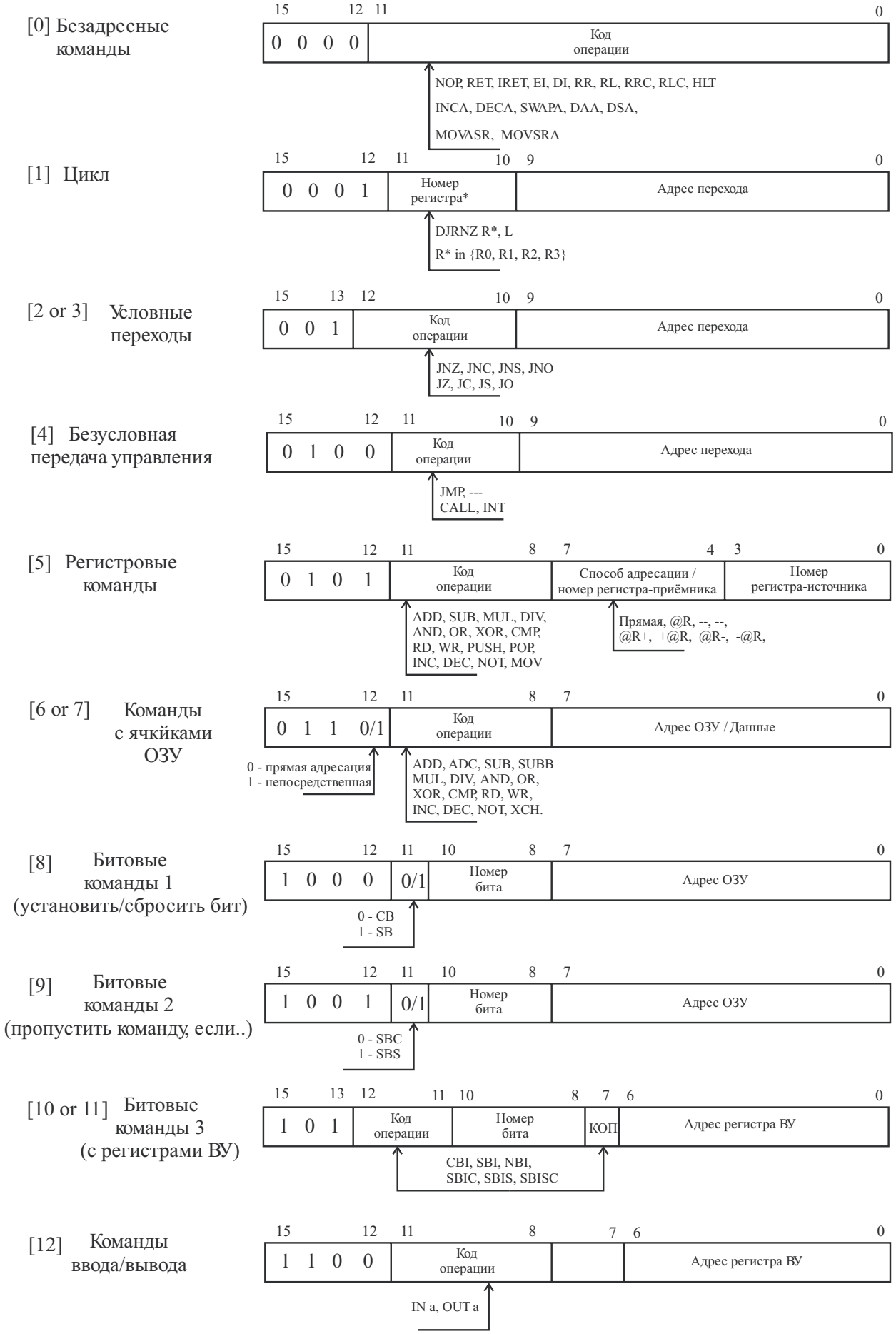

Рисунок 1. Форматы команд fN8

1.2. Таблица команд

В таблице приняты следующие обозначения:

Acc – содержимое аккумулятора;

DD – содержимое ячейки памяти или непосредственный операнд;

R – содержимое регистра общего назначения R;

R* – содержимое регистра или косвенно адресуемой через регистр ячейки памяти;

p – префикс перед именем регистра, определяющий способ адресации;

A – адрес ячейки памяти данных;

M(A) – содержимое ячейки памяти данных по адресу A;

C – флаг PSW[4] переноса(заёма)

SPL – содержимое указателя стека;

L – адрес перехода (метка или абсолютный);

CR – содержимое регистра команд;

RIO - содержимое регистра внешнего устройства;

a - номер (адрес) регистра внешнего устройства;

b - номер бита в байте;

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги O	C	A	N	Z
Безадресные команды [Код 0]							
NOP	Нет операции	PCL := PCL + 1	-	-	-	-	-
RET	Возврат из подпрограммы	PCL := M(SS.SPL); Inc(SPL); CS := M(SS.SPL)[1:0]; Inc(SPL)	-	-	-	-	-
IRET	Возврат из прерывания	PCL := M(SS.SPL); Inc(SPL); PSW[5:0].CS := M(SS.SPL); Inc(SPL)	-	-	-	-	-
EI	Разрешить прерывание	FI := 1	-	-	-	-	-
DI	Запретить прерывание	FI := 0	-	-	-	-	-
RR	Сдвиг аккумулятора правый циклический	Acc[7:0] := Acc[0].Acc[7:1]; FC := Acc[0]	-	-	-	-	-
RL	Сдвиг аккумулятора левый циклический	Acc[7:0] := Acc[6:0].Acc[7]; FC := Acc[7]	-	-	-	-	-
RRC	Сдвиг аккумулятора правый через перенос	Acc[7:0] := FC.Acc[7:1]; FC := Acc[0]	-	-	-	-	-
RLC	Сдвиг аккумулятора левый через перенос	Acc[7:0] := Acc[6:0].FC; FC := Acc[7]	-	-	-	-	-

Смотри продолжение на следующей странице

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги O	C	A	N	Z
NOTA	Инверсия аккумулятора	$Acc := \overline{Acc}$	0	0	0	+	+
INCA	Инкремент аккумулятора	Acc := Acc + 1	-	+	+	+	+
DECA	Декремент аккумулятора	Acc := Acc – 1	-	+	+	+	+
SWAPA	Обмен тетрадами аккумулятора	Acc[7:4] ⇔ Acc[3:0]	-	-	-	-	-
DAA	Десятичная коррекция сложения	См. раздел 1.3.1	-	+	+	+	+
DSA	Десятичная коррекция вычитания	См. раздел 1.3.1	-	+	+	+	+
MOVSP	Загрузка SPL	SPL := Acc;	-	-	-	-	-
MOVAPSW	Прочитать PSW	Acc := PSW	-	-	-	-	-
MOVASR	Прочитать SR	Acc := SR	-	-	-	-	-
MOVSRA	Загрузить SR	SR := Acc	-	-	-	-	-
HLT	Стоп	Прекратить командные циклы	-	-	-	-	-
Цикл [Код 1]							
DJRNZ R^c, L	Цикл	$R^c := R^c - 1$; if $R^c \neq 0$ then goto L	-	-	-	-	-
$R^c \in \{R0, R1, R2, R3\}$							
Команды условных переходов [Код 2 or 3]							
JNZ L	Переход, если не ноль	if Z=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JNC L	Переход, если не перенос	if C=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JNN L	Переход, если не отрицательно	if N=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JNO L	Переход, если не переполнение	if O=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JZ L	Переход, если ноль	if Z=1 then CS.PCL := CR[9:0]	-	-	-	-	-
JC L	Переход, если перенос	if C=1 then CS.PCL := CR[9:0]	-	-	-	-	-
JN L	Переход, если отрицательно	if N=1 then CS.PCL := CR[9:0]	-	-	-	-	-
JO L	Переход, если переполнение	if O=1 then CS.PCL := CR[9:0]	-	-	-	-	-

Смотри продолжение на следующей странице

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги O C A N Z
Команды безусловной передачи управления [Код 4]			
JMP L	Безусловный переход	CS.PCL := CR[9:0]	- - - - -
CALL L	Вызов подпрограммы	Dec(SPL); M(SS.SPL)[1:0] := CS; Dec(SPL); M(SS.SPL) := PCL; CS.PCL := CR[9:0]	- - - - -
INT v	Вызов прерывания	Dec(SPL); M(SS.SPL) := PSW[5:0].CS; Dec(SPL); M(SS.SPL) := PCL; PCL := M(2v); CS := M(2v+1)[1:0]; FI := 0	- - - - -
v = CR[2:0] – вектор прерывания			
Регистровые команды [Код 5] (Для команд ADC и SUBB – [Код F])			
ADD pR	Сложение	Acc := Acc + R*	+ + + + +
ADC pR	Сложение с переносом	Acc := Acc + R* + C	+ + + + +
SUB pR	Вычитание	Acc := Acc – R*	+ + + + +
SUBB pR	Вычитание с заёмом	Acc := Acc – R* – C	+ + + + +
MUL pR	Умножение	R1.R0 := Acc × R*	- - - - -
DIV pR	Деление	Acc := Acc : R*	- - - - +
AND pR	Конъюнкция	Acc := Acc & R*	0 0 0 + +
OR pR	Дизъюнкция	Acc := Acc ∨ R*	0 0 0 + +
XOR pR	Неравнозначность	Acc := Acc ⊕ R*	0 0 0 + +
CMP pR	Сравнение	R* – Acc	+ + + + +
RD pR	Чтение	Acc := R*	- - - - -
WR pR	Запись	R* := Acc	- - - - -
XCH pR	Обмен	R* ⇔ Acc	- - - - -
PUSH R	Поместить в стек	Dec(SPL); M(SPL) := R;	- - - - -
POP R	Извлечь из стека	R := M(SPL); Inc(SPL)	- - - - -
INC R	Инкремент	R := R + 1;	- + + + +
DEC R	Декремент	R := R – 1;	- + + + +
NOT R	Инверсия	$R := \overline{R}$;	0 0 0 + +
MOV R_r, R_t	Копирование	$R_r := R_t$	- - - - -

Смотри продолжение на следующей странице

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги O	C	A	N	Z
Команды с ячейками ОЗУ [Код 6 or 7] Код 6 – прямая адресация (A), 7 – непосредственная (#A)							
ADD A	Сложение	Acc := Acc + DD	+	+	+	+	+
ADC A	Сложение с переносом	Acc := Acc + DD + FC	+	+	+	+	+
SUB A	Вычитание	Acc := Acc – DD	+	+	+	+	+
SUBB A	Вычитание с заёмом	Acc := Acc – DD – FC	+	+	+	+	+
MUL A	Умножение	R1.R0 := Acc × DD	-	-	-	-	-
DIV A	Деление	Acc := Acc : DD	-	-	-	-	+
AND A	Конъюнкция	Acc := Acc & DD	0	0	0	+	+
OR A	Дизъюнкция	Acc := Acc ∨ DD	0	0	0	+	+
XOR A	Неравнозначность	Acc := Acc ⊕ DD	0	0	0	+	+
CMP A	Сравнение	DD – Acc	+	+	+	+	+
RD A	Чтение	Acc := DD	-	-	-	-	-
WR A	Запись	M(A) := Acc	-	-	-	-	-
XCH A	Обмен	M(A) ⇔ Acc	-	-	-	-	-
INC A	Инкремент	M(A) := M(A) + 1	-	+	+	+	+
DEC A	Декремент	M(A) := M(A) – 1	-	+	+	+	+
NOT A	Инверсия	M(A) := $\overline{DD}$	0	0	0	+	+
Битовые команды 1 [Код 8]							
CB A,b	Сбросить бит	M(A)[b] := 0	-	-	-	-	-
SB A,b	Установить бит	M(A)[b] := 1	-	-	-	-	-
Битовые команды 2 [Код 9]							
SBC A,b	Пропустить команду, если бит сброшен	if M(A)[b]=0 then PCL := PCL + 1	-	-	-	-	-
SBS A,b	Пропустить команду, если бит установлен	if M(A)[b]=1 then PCL := PCL + 1	-	-	-	-	-
Битовые команды 3 [Код A or B]							
CBI a,b	Сбросить бит	RIO(a)[b] := 0	-	-	-	-	-
SBI a,b	Установить бит	RIO(a)[b] := 1	-	-	-	-	-
NBI a,b	Инвертировать бит	$RIO(a)[b] := \overline{RIO(a)[b]}$	-	-	-	-	-
SBIC a,b	Пропустить команду, если бит сброшен	if RIO(a)[b]=0 then PCL := PCL + 1	-	-	-	-	-
SBIS a,b	Пропустить команду, если бит установлен	if RIO(a)[b]=1 then PCL := PCL + 1	-	-	-	-	-

Смотри продолжение на следующей странице

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги O C A N Z
SBISC a,b	Пропустить команду, если бит установлен и сбросит бит	if RIO(a)[b]=1 then (PCL := PCL + 1; RIO(a)[b] := 0)	- - - - -
Команды ввода/вывода [Код C]			
IN a	Ввод	Acc := RIO(a)	- - - - -
OUT a	Вывод	RIO(a) := Acc	- - - - -

1.3. Пояснения к некоторым командам таблицы 1

1.3.1. Безадресные команды десятичной коррекции

DAA – десятичная коррекция байта в Acc после сложении (кодировка «8421»)

$$\text{if } (B_L > 9) \text{ or } (AC) \text{ then } B := B + 0x06; \tag{1}$$

$$\text{if } (B_H > 9) \text{ or } (C) \text{ then } B := B + 0x60; \tag{2}$$

DSA – десятичная коррекция байта в Acc после вычитании (кодировка «8421»)

$$\text{if } (AC) \text{ then } B := B - 0x06; \tag{3}$$

$$\text{if } (C) \text{ then } B := B - 0x60; \tag{4}$$

Здесь B – предварительный результат операции сложения (вычитания), B_H – старшая тетрада B, B_L – младшая тетрада B, AC – перенос (заём) из младшей тетрады, C – перенос (заём) из байта.

1.3.2. Умножение

Содержимое аккумулятора можно умножить на содержимое регистра, прямо или косвенно адресуемой ячейки памяти или 8-разрядную константу. Формат произведения – два байта, поэтому оно размещается не в аккумуляторе, а в регистрах R1 и R0, причём в R1 размещается старший байт произведения, а в R0 – младший.

1.3.3. Деление

Целочисленное деление: байт Acc делится на байт делителя, целая часть частного помещается в Acc. Если содержимое Acc меньше делителя, то результат деления равен 0, устанавливается флаг $Z = 1$. В случае ненулевого результата деления флаг $Z = 0$, остальные флаги в операции деления не изменяются. В качестве делителя можно использовать непосредственный операнд, содержимое регистра, содержимое прямо или косвенно адресуемой ячейки памяти.

1.3.4. Цикл

DJRNZ R,L – декремент регистра и проверка. Если после декремента содержимое регистра $\neq 0$, то осуществляется переход по указанному адресу, иначе – на следующую команду. Можно использовать для организации циклов. Работает только с регистрами R0, R1, R2, R3.

1.3.5. Команды безусловной передачи управления

CALL L – вызов подпрограммы по прямому адресу. Использует две ячейки стека для размещения 10-разрядного адреса возврата (в старшем байте занято только два младших разряда).
INT v – вызов обработчика прерывания по вектору v. Как и команда CALL, сохраняет адрес возврата, а в свободные 6 разрядов старшего байта помещает флаги I, O, C, AC, N, Z. Таблица векторов прерываний в ОЗУ начинается с адреса 0x000, вектор прерываний $v \in \{0, 1, \dots 7\}$. Адрес обработчика прерываний загружается из ячеек с адресами (2v) и (2v+1)[1:0] в PCL и SR[1:0] соответственно.

2. Директивы компилятора

При написании программы на языке Ассемблер можно пользоваться следующими директивами компилятора:

.c <сегмент> – выбирает текущий сегмент компиляции;

.org <адрес> – изменяет текущий адрес компиляции;

.db <байт, байт, ... байт> – загрузка констант размером в 1 байт в текущий сегмент начиная с текущего адреса компиляции;

.dw <слово, метка, ... слово> – загрузка констант размером в 2 байта в текущий сегмент начиная с текущего адреса компиляции.

2.1. Загрузка констант

В процессе компиляции возможна загрузка констант в сегмент данных (например, таблицы ASCII-кодов символов). По умолчанию сегменты «0» и «1» отведены для кода, сегмент «2» – для данных, а сегмент «3» – для стека. Если (лучше в конце текста программы) поставить директиву **.c 2**, то компилятор продолжит компиляцию фактически в сегмент данных. Например, если требуется разместить таблицу 7-сегментных кодов[1] десятичных цифр в сегменте данных начиная с адреса 0x90, то в конце текста программы можно добавить такой фрагмент:

$.c\ 2$
$.org \quad 0x90$
$.db \quad 0x3F,\ 0x06,\ 0x5B,\ 0x66,\ 0x6D,\ 0x7D,\ 0x07,\ 0x7F,\ 0x6F,\ 0x77$

2.2. Генерация таблицы векторов прерываний

Использование директив позволяет изменить установленную по умолчанию точку старта программы – 0x10, например, на адрес 0x20 (a). Ещё проще поставить в нужном месте метку, например, *Start:* и объявить её точкой старта (b).

a)	*b)*
$.org \quad 0$	$.org \quad 0$
$.db \quad 0x20, \quad 0x0$	$.dw \quad Start$

Эти же директивы позволяют автоматизировать заполнение таблицы векторов прерываний. Таблица векторов прерываний в fN8 размещается в младших адресах памяти 0x000 – 0x00F, каждый вектор занимает два байта. Вектор 0 определяет точку старта (по умолчанию — 0x010), вектора 1, 2, 3 и 4 по умолчанию присваиваются клавиатуре, таймеру-2, таймеру-5 и контроллеру 7-сегментной индикации соответственно. Остальные вектора (5 — 7) можно использовать для других разрабатываемых внешних устройств или для подключения нескольких экземпляров одинаковых ВУ с разными векторами. Адрес младшего байта вектора определяется как его удвоенный номер.

[1] В этом примере сегменту **A** индикатора соответствует младший бит кода.

Если поставить метки в начале всех обработчиков прерываний, то загрузка векторов прерываний может выглядеть следующим образом (для случая, когда вектора определяются последовательно):

.org 2

.dw IntKey, IntTim2, IntTim5, Int7Seg

По адресам неиспользуемых векторов в таблице векторов прерываний сохраняются значения 0. Если в системе используются те ВУ, вектора прерываний которых по умолчанию следуют не подряд, то можно или при подключении поменять им вектора в желаемом порядке или в директиве заполнения таблицы по неиспользуемым адресам записывать 0, например:

.org 2

.dw IntKey, 0, 0, Int7Seg ,

если подключать только контроллер клавиатуры и контроллер 7-сегментной индикации и оставить им назначаемые по умолчанию вектора.