**I.** Язык **A-Natural**

|  |  |  |
| --- | --- | --- |
| I - 1 | **As.80** | Введение в A-Natural |
| I - 4 | **Синтаксис** | Правила для A-Natural |
| I - 7 | **Определения** | Определения символов, данных и кода |
| I - 10 | **Перемещения** | О перемещении данных |
| I - 13 | **Арифметика** | Обработка данных на 8080 |
| I - 17 | **Управление** | Переходы и управление процессом |
| I - 19 | **Методы** | 8080 для взрослых |
| I - 24 | **Резюме** | Краткое описание A-Natural |

**ИМЯ**

**As.80** - Введение в A-Natural

**ФУНКЦИЯ**

A-Natural, также известный как As.80, является описательным языком ассемблера для использования с компьютерами Intel 8080, Intel 8085 или Zilog Z80. Он имеет единое соглашение об именовании регистров, краткую запись операторов для большинства машинных инструкций и правила форматирования, которые делают логические группировки очень заметными. Тем не менее, язык предоставляет программисту всю мощь базового набора команд и старается не создавать неожиданных побочных эффектов, таких как изменение неназванных регистров.

Контраст между A-Natural и языком ассемблера может быть достигнут с помощью кодовой последовательности для вычитания двух 16-разрядных целых чисел в ячейке памяти X и в 4 + de:

a = \*(bc = &X) - \*(hl = 4 + de) -> \*bc = \*(bc + 1) -^ \*(hl + 1) -> \*bc

или, в ассемблере:

LXI B,X INX B

LDAX B UJAX B

LXI H,4 INX H

DAD D SBB M

SUB M STAX B

STAX B

Обратите внимание, что регистр a имеет явное имя, все ссылки на hl используют одно и то же имя (не H, M или неявное), что все перемещения данных вызываются операторами присваивания "=" и "->", и что используются несколько операторов для обозначения режимов адресации ("\*", "&") и арифметики ("-", "-^").

Для чтения строки A-Natural требуется повторное применение нескольких правил:

1. Считывая слева направо, найдите первый двоичный оператор, в этом случае "="
2. Если операнд слева не является простым термином, вычислите подвыражение. В этом случае операнд является просто a.
3. Если операнд справа не является простым термином, оцените подвыражение. Здесь \*(bc = &X) требует создания кода для bc = &X, который является "LXI B,X", а затем сводится к \*bc.
4. Произведите код для двоичного оператора, используя два приведенных операнда, здесь a = \*bc.
5. Оставьте левый операнд, отбросьте остальные и повторите. Затем линия становится

a - \*(hl = 4 + de) -> \*bc = \*(bc + 1) -^ \*(hl + 1) -> \*bc

Согласно этой схеме все двоичные операторы имеют одинаковый приоритет, поэтому выражения читаются строго слева направо, за исключением тех случаев, когда порядок явно изменяется скобками.

Использование A-Natural описывается следующими подразделами:

|  |  |  |
| --- | --- | --- |
| **As.80** | **-** | вы смотрите его. |
| **Синтаксис** | **-** | как формировать идентификаторы, константы, выражения и т.д. |
| **Определения** | **-** | придание значений символам и выражениям, генерация констант. |
| **Ходы** | **-** | регистры 8080 и основные перемещения данных. |
| **Арифметика** | **-** | манипуляции с байтами и адресами, а также более причудливые ходы. |
| **Управление** | **-** | управление потоком и управление процессом. |
| **Методы** | **-** | как иметь дело с целыми числами и адресами, как смешивать их с кодом на языке C. |
| **Резюме** | **-** | язык в двух словах. |

**СМ. ТАКЖЕ**

Очень полезный набор подпрограмм, библиотека машинного интерфейса 8080, описан в другой части этого руководства. Эти процедуры используются компилятором C для упрощения ввода/вывода функций, для выполнения целочисленных операций умножения, деления и сдвига, а также для выполнения всех операций с длинными, плавающими и двойными числами. Использование этой библиотеки от A-Natural может существенно сэкономить усилия при кодировании и снизить требования к памяти при смешивании A-Natural с кодом на языке C.

Автономное использование транслятора A-Natural описано на странице руководства по программе anat. Ее использование в качестве постпроцессора для компилятора 8080 C описано на странице руководства для C. Обе эти страницы включены в руководство по системному интерфейсу вместе с подпрограммами, используемыми для связи с системой.

Язык C и стандартная библиотека вызываемых функций Си описаны в Руководстве пользователя компилятора C.

**ПРИМЕР**

Вот полный текст библиотечного модуля в A-Natural:

/ imul

/ copyright (c) 1979 by Whitesmiths, Ltd.

/ mul int into int

/ stack: P, bc, hl, pc, R, L

/

/ P = 0

/ while (R)

/ if (R & 1)

/ P =+ L

/ L =<< 1

/ R =>> 1

/ L = P

public c.imul

R := 8

L := 10

c.imul:

sp <= hl <= bc <= (hl = 0)

.1:

a = \*(hl = R[1] + sp) | \*(hl - 1) jz .3

a = \*hl <^> -1 jnc .2

bc =^ (hl = L + sp)

hl <= sp + bc => sp

.2:

a = \*(hl = L + sp) + a -> \*hl = \*(hl + 1) +^ a -> \*hl

a = \*(hl = R[1] + sp) <-> -1 -> \*hl

a = \*(hl - 1) <^> -1 -> \*hl

jmp .1

.3:

bc <= sp ->^ (hl = L[-2] + sp)

jmp c.lret

**ИМЯ**

Синтаксис - правила написания A-Natural

**ФУНКЦИЯ**

На самом нижнем уровне программа A-Natural представлена в виде текстового файла, состоящего из строк, каждая из которых заканчивается точкой с запятой ';' новой строкой '\n' или комментарием. Комментарий состоит из всех символов между '/' и следующей новой строкой, включительно.

Текстовые строки разбиваются на токены, строки символов, возможно, разделенные пробелами. Пробел состоит из одного или нескольких непечатаемых символов, таких как пробел или табуляция. Его единственным эффектом является разграничение маркеров, которые в противном случае могли бы быть объединены. Токены принимают несколько форм:

**Идентификатор** - состоит из буквы, точки "." или подчеркивания "\_", за которыми следует ноль или более букв, точек, подчеркиваний или цифр. Заглавные буквы отличаются от строчных. При сравнении идентификаторов важно не более девяти символов. Более жесткие ограничения могут быть наложены на идентификаторы внешним миром. Существует также ряд идентификаторов, зарезервированных в качестве ключевых слов:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| . | cm | de | jmp | public | rst1 |
| .data | cmc | di | jnc | rc | rst2 |
| .text | cnc | e | jnz | ret | rst3 |
| a | cnz | ei | jp | rm | rst4 |
| af | cp | h | jpe | rnc | rst5 |
| b | cpe | hl | jpo | rnz | rst6 |
| bc | cpo | hlt | jz | rp | rst7 |
| c | cz | in | l | rpe | rz |
| call | d | lc | nop | rpo | sp |
| cc | daa | jm | out | rst0 | stc |

**Числовая константа** - состоит из десятичной цифры, за которой следует ноль или более букв и цифр. Если начальными символами являются "0x" или "0X", константа является шестнадцатеричным литералом и может содержать буквы "a" - "f", в любом случае, чтобы представить значения цифр от 10 до 15 соответственно. В противном случае начальный "0" означает восьмеричный литерал, который, тем не менее, может содержать цифры "8" и "9". Ненулевая ведущая цифра подразумевает десятичный литерал. Числа представлены с точностью до 16 бит. Переполнение не диагностируется.

**Символьная константа** - состоит из одинарной кавычки, за которой следует ноль или более символьных литералов, за которыми следует одинарная кавычка. Символьный литерал состоит из а) любого символа, кроме '\', новой строки или одинарной кавычки, значение которого является представлением этого символа в кодировке ASCII; б) символ '\', за которым следует любой символ, кроме новой строки, значение которого является представлением этого символа в кодировке ASCII, за исключением символов в последовательности <'b', 't', 'v', 'f', 'n', 'r', '(', '!', ')', '^'> в любом случае имеют значения ASCII для соответствующих членов последовательности <backspace, горизонтальная табуляция, вертикальная табуляция, подача формы, новая строка, возврат каретки , '{', '|', '}', '~'> или в) '\', за которым следуют от одной до трех десятичных цифр, значение - восьмеричное число, представленное этими цифрами. Внутри кавычек нельзя использовать новую строку. Значение константы - это целое число с основанием 256, цифры которого являются значениями символьных литералов. Сохраняются только две последние такие цифры. Переполнение не диагностируется.

**Строковая константа** - это точно так же, как символьная константа, за исключением того, что двойные кавычки '"' используются для разграничения строки. Строка - это сокращение для ряда байтовых констант, значения которых являются символьными литералами в строке.

**Пунктуация** - состоит из заранее определенных строк от одного до трех символов. Полный набор знаков препинания для A-Natural таков:

! + \_^ <+> = [

& +^ : <-> =! ]

( - :: <= => ^

) ->> := <><> =а^ |

\* ->^ <\*> <^> =^ ,

Подбирается максимально длинная строка пунктуации, так что "=!", например, всегда распознается как "=!", и никогда как '=', за которым следует '!'. Операторы "!" и "|" эквивалентны.

**Другие символы**, такие как только "@" или "\", являются незаконными вне символьных или строковых констант и диагностируются.

Ничто другое не является токеном в A-Natural.

Для тех, кто любит форму Бэкуса-Наура, A-Natural может быть кратко выражен произведениями:

program := <NULL>

| program line

line := <NULL> EOL

| term EOL

| STRING EOL

| expr EOL

| IDENT : линия

expr := term BINOP term

| expr BINOP term

term := NUMBER

| IDENT

| UNOP term

| IDENT : term

| IDENT := expr

| ( expr )

| term [ expr ]

где получаются следующие определения:

<NULL> это ничто

EOL это терминатор строки, например ';' или '\n'

STRING это строковая константа

BINOP это один из нескольких бинарных операторов которые будут описаны

NUMBER это числовая константа или символьная константа

IDENT это идентификатор

UNOP это один из нескольких унарных операторов подлежащих описанию

Проще говоря, это означает, что **программа** - это объединение одной или нескольких строк, каждая из которых может быть пустой, или содержать термин, или содержать строковую константу, или содержать выражение. Строке может предшествовать ноль или более меток, где метка - это идентификатор, за которым следует двоеточие.

**Выражение** - это последовательность из двух или более членов s, разделенных двоичными операторами. Выражения читаются слева направо. **Термин**, в самом простом виде, состоит только из числовой константы, символьной константы или идентификатора. Однако он может также состоять из определения, которое является идентификатором, за которым следует ":=" и выражение, или он может состоять из выражения в круглых скобках. Ему могут предшествовать ноль или более меток и/или унарных операторов; за ним могут следовать ноль или более индексов, каждый из которых заключен в квадратные скобки.

Эта кажущаяся общность ограничена несколькими важными семантическими ограничениями. Эти ограничения обсуждаются по мере того, как каждая из интересных постановок посещается более подробно.

**ОШИБКИ**

m должен быть зарезервированным идентификатором.

**ИМЯ**

**Определения** - Определение символов, данных и кода

**ФУНКЦИЯ**

Цель A-Natural состоит в том, чтобы создать узел операторов ассемблера или перемещаемых двоичных файлов, который определяет некоторые глобально известные символы, определяет начальное содержание областей данных и выражает последовательности инструкций, формирующих тело каждой функции. Важной подцелью является определение значения и атрибутов терминов и внутренне используемых символов.

Символы и термины обозначают элементы данных, которыми нужно манипулировать. Каждый из них можно рассматривать как рецепт определения местоположения данных, о которых идет речь. Существует ряд предопределенных идентификаторов для разговоров о регистрах, в то время как числовые значения представлены числовыми и символьными константами, а также строками. Все адреса памяти и ссылки берутся из нескольких источников.

Существует заранее определенный символ .text, который обозначает следующее место для генерации кода в разделе памяти program text (только для чтения). Символ .data обозначает следующее место для генерации кода в разделе data (writeable) памяти. Третий символ, записываемый просто ., - это следующее место, в которое A-Natural будет генерировать код, .text будь то текст или данные ..

Пока еще неопределенный идентификатор может появиться в определяющем операторе форм

x := 3

Отныне любая ссылка на x будет приниматься как ссылка на литерал номер 3, однобайтовую константу. Правая сторона определяющего оператора ":=" должна быть выражением, которое вычисляет число, адрес памяти или ссылку на память.

Обычный источник ссылок на память ., как в:

M := .

Это определяет M как содержимое следующего байта (или двух байтов) памяти. Это происходит так часто, что предусмотрена специальная нотация метки:

M:

имеет тот же эффект, определяя М как ., но короче в написании и может сопровождаться дополнительными метками или термином.

Если необходимо говорить собственно об адресе памяти, а не о содержимом памяти, то оператор "адрес памяти" &M преобразует ссылку на память M в двухбайтовую константу, значением которой является адрес M. Аналогично, &3 преобразует однобайтовую константу в двухбайтовую, в то время как & & 3 оставляет &3 без изменений.

Для преобразования двухбайтовой константы в ссылку на память используется унарный оператор "косвенный on" \*. Таким образом, \*&M - это то же самое, что и M, а \*&3 - это ячейка памяти по абсолютному адресу 3. Оператор "\*" также используется в сочетании с некоторыми двухбайтовыми регистрами для указания косвенной ссылки на регистр.

Другими унарными операторами являются "+", "-" и "!", которые изменяют значение одной или двухбайтовых абсолютных констант. Значение оставляется в покое "+", отрицается "-" и дополняется "!". Оператор public делает свой операнд, который должен быть символом, известным во всех файлах во время загрузки:

public x

Этот термин не делает ничего, кроме объявления x глобально известным.

Значение термина может быть изменено постоянным смещением с помощью подстрочной нотации M[5]. Этот термин относится к местоположению на пять байт выше M, в то время как 3[5] - это однобайтовая константа 8, а &3[5] - двухбайтовый абсолютный адрес 8.

Генерация кода использует эту нотацию плюс несколько дополнительных понятий. Однобайтовая константа, стоящая отдельно, генерирует один байт кода, имеющего это значение:

N: 0

Байтовое расположение N инициализируется нулем.

Для генерации двухбайтовой константы:

список: &0

или:

список: &head

где голова - это, по-видимому, еще одна метка. По специальному разрешению, & может быть опущено в этом контексте.

Строки могут использоваться для указания последовательности байтов. Например:

"help\n\0"

это то же самое, что

'n'; 'e'; 'l'; 'p'; '\n'; 0

только гораздо проще печатать.

Конкатенация - это единственная операция, разрешенная для строк. Оператор запятой позволяет объединять однобайтовые константы, двухбайтовые абсолютные константы и строки для формирования более длинных строк, как в:

"help", "\n", 0

'h', 'e', "lp\n\0"

и то, и другое эквивалентно приведенной выше строке.

Последний унарный оператор-это своеобразный оператор, называемый "литералом". Термин =&M приводит к тому, что двухбайтовый литерал, значение которого является адресом M, запоминается в таблице. В конце исходного ввода) генерация кода возвращается к .text и генерируется таблица; идентичные литералы объединяются как можно больше. Значение термина =&M - это ссылка памяти на тот литерал, который должен быть испущен. Это позволяет манипулировать перемещаемыми адресами по байту за раз с помощью ссылок на (=&M)[0] и (=&M)[1].

Строковые литералы также могут быть сгенерированы путем записи таких строк, как

hl = &="help\n\0" => sp

это удобный способ определить литеральную строку и сразу поместить ее адрес в стек. Например, длинная константа может быть записана либо в виде строки из четырех байт, либо в виде двух двухбайтовых констант:

hl = &=(-1, -4) => ип

чтобы получить указатель на длинный -4 в стеке.

Переключиться с генерации кода в .text (что всегда начинает делать A-Natural) на генерацию данных в .data:

. := .data

и переключиться обратно:

. := .text

и зарезервировать место, как для массива:

. := .[50] / зарезервировать 50 байт

Обратите внимание, что . - это единственный идентификатор, который может быть переопределен, что его переопределение должно быть относительно .text или .data и что . не может быть скопировано поверх уже созданного кода.

Вся остальная генерация кода определяется двоичными операторами между терминами, как описано в подразделе As.80 данного руководства. Таким образом,

X op Y op 7.

это стенография для

X op Y; X op Z /x простой термин

и приводит к производству кода для 1) термы X, 2) термина Y, 3) X op Y, 4) термина Z и 5) X op Z.

В последующих разделах этого руководства описываются определенные бинарные операторы, комбинации терминов, которые они принимают, и выполняемые действия.

**ИМЯ**

**Перемещения** - Как переместить данные о

**ФУНКЦИЯ**

Intel 8080 - это компьютер, который имеет дело с 8-битными байтами и 16-битными адресами (словами) с использованием арифметики двойного дополнения. Все инструкции и байты данных извлекаются из одного и того же адресного пространства размером 65 536 байт, которое может представлять собой смесь ПЗУ (для кода и таблиц) и ОЗУ (для стека и данных). Слова хранятся сначала менее значимым байтом. Вход/выход - это байт за раз из отдельного 256-портового адресного пространства.

В байтовых операциях могут участвовать семи-байтовые регистры. Они называются a (который часто служит накопителем байтов), b, c, d, e, h и l, Другой байтовый регистр, называемый f, содержит флаги, которые устанавливаются или тестируются как побочное влияние некоторых инструкций. Эти флаги расположены в порядке убывания значимости:

|  |  |  |
| --- | --- | --- |
| f7 (S) | - | производится в соответствии со знаком результата большинства байтовых арифметических инструкций. |
| f6 (Z) | - | устанавливается, если результат равен нулю для большинства байтовых арифметических инструкций. |
| f5 (0) | - | всегда 0. |
| f4 (AC) | - | устанавливается на перенос от бита 3 до бита 4 в большинстве байтовых арифметических инструкций. |
| f3 (0) | - | всегда 0. |
| f2 (P) | - | устанавливается, если результат имеет четное число из одного бита в большинстве байтовых арифметических инструкций. (Обратите внимание, этот флаг также служит индикатором переполнения на z80, единственном известном недостатке его восходящей совместимости с 8080.) |
| f1 (1) | - | всегда 1. |
| f0 (с) | - | устанавливается на перенос из бита 7 байта многих арифметические инструкции, устанавливается на перенос из разрядной 15 на слово добавляет, участвует в байт вращения, и может быть задан явно или дополняются. Обычно его называют флагом переноса. |

Настройки флагов будут обсуждаться с каждой инструкцией, которая их использует.

Эти байтовые регистры объединяются в пары, образуя четыре регистра слов, называемых af, bc, de и hl. В каждом случае первый регистр с именем содержит более значимый байт, hl чаще всего используется в качестве аккумулятора указателя, af можно только толкать или выскакивать.

Другим регистром слов является sp, который выделен в качестве указателя стека. Только количество слов может быть перемещено в стек или из него, хотя нет требования, чтобы sp содержал четный адрес. Количество выталкивается путем уменьшения sp на два, затем используя sp в качестве адреса хранения менее значимого байта; popping сначала использует sp в качестве адреса источника, а затем добавляет два к sp. Таким образом, sp всегда указывает на последнее слово, помещенное в стек, то есть на вершину стека.

Последний регистр слов - это pc, который выделен в качестве счетчика программ. В начале каждой команды pc указывает на opcode байт кода операции, за которым может следовать однобайтовый непосредственный операнд или двухбайтовый адрес. Для изменения pc и, следовательно, изменения потока управления используется специальный класс инструкций; они называются переходами, вызовами, возвратами и сбросами, pc не является зарезервированным идентификатором в A-Natural.

Любой байтовый регистр может быть скопирован в другой, как в:

e = b

или эквивалентно,

b -> c

который копирует b в c. Более того, любая операция , допустимая для b, c, d, e, h и l , также допустима для байта, на который указывает hl, то есть \*hl. Таким образом,

a = \*hl -> c

вызывает a загрузку a с копией байта, на который указывает hl, а затем копирование в c. Эти перемещения байтов не влияют ни на какие флаги.

Отметим, что восемь специальных оазисов

b = b; c = c; ...; \*hl = \*hl

этого следует избегать, потому что инструкция либо ничего не делает, либо останавливает компьютер (\*hl = \*hl). Однобайтовая операция no лучше всего кодируется следующим образом

ноп

в то время как остановка лучше всего написана

hlt

Идентификаторы nop и hit просто определяются как небольшие числа, имеющие соответствующие секретные значения; поскольку указание только числа приводит к тому, что байт с этим значением генерируется встроенно, создаются соответствующие инструкции.

Можно присвоить байтовую константу любому байтовому регистру, или \*hl, записав

d = 13

\*hl = 0200

Первая строка копирует десятичное число 13 в d, вторая-восьмеричное число 200 в байт, на который указывает hl.

Регистрация а может участвовать и в нескольких других ходах:

a = \*bc; a -> \*bc

a = \*de; a -> \*de

a = M; a -> M

Здесь M обозначает произвольный адрес памяти загружаемого или сохраняемого байта.

Значения слов могут быть присвоены большинству регистров слов:

bc = &1024

de = &M

hl = 3

sp = &стек

Третья строка-это просто стенография

hl = 43

в то время как четвертый используется очень редко, так как sp обычно настраивается только один раз для каждой программы. Гораздо более часто используемым назначением является;

sp = hl

после вычисления модифицированной версии sp в hl. Обратите внимание, что af не может быть назначен; назначение ПК происходит под видом прыжков, вызовов и т. Д.

Также допустимо загружать и хранить hl непосредственно из памяти:

hl = X -> Y

Это копирует слово из памяти в точке X в память в точке Y, используя hl, и его не следует путать с загрузкой непосредственного значения , как показано выше.

Наконец, можно произвольно копировать слова между bc, deи hl, написав такие строки , как:

bc = de

hl = bc

Эти операции фактически выполняются по байту за раз, и поэтому их можно рассматривать либо как двухбайтовые инструкции, либо как сокращение для таких строк, как:

c = e; b = d

l = c; h = b

Разница редко бывает существенной.

**ИМЯ**

Арифметика - Манипулирование данными на 8080

**ФУНКЦИЯ**

8080 манипулирует данными в виде восьмибитных байтов и (в гораздо меньшей степени) в виде 16-битных слов. Арифметика в обычном смысле этого слова ограничена сложением и вычитанием; нет никаких инструкций умножения или деления. Но есть также ряд логических операций, а также повороты различных вкусов, которые делают возможными вычисления с множественной точностью и более сложные операции. Поэтому все эти вещи свалены в одну кучу.

Легко добавить или вычесть единицу из любого байтового регистра или \*hl:

с + 1

\*hl - 1

Эти операции устанавливают все, кроме флага переноса. С другой стороны, операции инкремента и декремента слов:

bc + l, de + 1, hl + 1, sp + 1

bc - 1, de - 1, hl - 1, sp - 1

- никаких флагов вообще. Кстати, увеличение или уменьшение sp редко бывает полезным.

Любой из байтовых регистров, \*hl или байтовая константа могут быть добавлены к:

a + c + 060

Каждое добавление влияет на все флаги. Добавление слов может происходить только от bc, deили sp до hl:

hl = 4 + sp -> bc

Это лучший способ получить sp + 4 в bc, используя hl в качестве аккумулятора слов. Затрагивается только флаг переноса.

Вся остальная арифметика В 8080 включает в себя байтовый операнд, который находится в aрегистре at \*hlили константе , и второй байтовый операнд в a, причем результат остается в a и все флаги затронуты. Полный набор операций, включая уже обсуждавшееся добавление, таков::

a + a / add

a +^ \*hl / add with carry

а - 3 / вычесть

a -^ b / вычитание с заимствованием

a & h / побитовая и

a ^ 1 / побитовое исключение или

a | a / побитовое включение или

a :: e / сравнить

Добавление a к a эффективно удваивает его или сдвигает его на одно место, одновременно перемещая предыдущий знак bit a7 в флаг переноса.

Add with carry - это то же самое, что add, за исключением того, что если флаг переноса установлен в начале инструкции, то единица добавляется к a вместе с правым операндом.

Вычитание добавляет дополнение двоек правого операнда к a, устанавливая флаг переноса, если "заимствование" должно распространяться на следующий более значительный байт. Вычитание с заимствованием продолжает эту цепочку, подобно тому, как сложение с переносом делает для сложения.

Побитовые операторы следуют стандартным таблицам истинности:

X Y X&Y X^Y X|Y

0 0 0 0 0

0 1 0 1 1

1 1 1 0 1

1 0 0 1 1

Эти операции безоговорочно очищают флаги carry и AC.

Обратите внимание, что a | a или a & a оставляет a без изменений, но устанавливает флаги S и Z для отражения его значения. Сравнение также оставляет a неизменным a, устанавливая флаги, чтобы отразить эффект вычитания правого операнда из a.

Оператор дополнения:

а =! а

заменяет a на его дополнение ones, не затрагивая никаких флагов, в то время как:

даа

jiggers up the a register after an add to give the correct result for adding decimal numbers packed as two four-bit digits per word. Это единственная инструкция, которая делает разумным использование флага AC; он устанавливает все флаги, чтобы десятичная сумма могла быть продолжена до более значительных байтов.

Регистр a может быть повернут на одно место влево или вправо с флагом переноса или без него между a7 и a0:

a <^> 1<^> / поворот влево с переносом

a <^> -1<^> / поворот вправо с переносом

a <\*> 1<\*> / left without carry

a <\*> -1<\*> / справа без переноски

Чтобы управлять флагом переноса напрямую:

увд / комплекты переносят

cmc / комплектация carry

a | a / очищает перенос (но также влияет на другие флаги)

Два других оператора выполняют вращение после принудительного переноса в известное состояние:

a <+> 1<+> / set carry, поворот влево

a <+> -1<+> / sat carry, поворот вправо

a <-> 1<-> / очистить перенос через a|a, повернуть влево

a <-> -1<-> / очистить перенос через a|a, повернуть вправо

Последний наиболее часто используется для осуществления сдвига вправо без распространения знака. Все повороты влияют на флаг переноса; если он не участвует в сдвиге, он по-прежнему установлен, как если бы он это сделал. Обратите внимание, что оператор "<-><->" сворачивает флаги в результате a|a.

8080 имеет некоторые особые инструкции для более специализированных ходов, таких как толчок стека и поппинг, а также обмены. В сочетании с некоторыми простыми арифметическими последовательностями они дают дополнительный набор более причудливых движений данных.

Чтобы протолкнуть регистр слов в стек:

sp

bo => sp / или de, или hl, или af

Любая форма имеет одинаковый эффект. Поппинг-это обратная нотация:

sp => bc / или de, или hl, или af

британская колумбия

Также можно обмениваться hl со словом верхнего стека:

hl <> \*sp

или

\*sp <> hl

Обратите внимание , что это \*sp, который заменяется на hl, а не sp. Единственный другой обмен

hl <> de

что также может быть написано в любом случае.

Единственный способ загрузить регистр слов из вычисленного адреса-это поместить адрес в hl и забрать по байту за раз, как в случае:

c = \*hl

hl + 1

b = \*hl

Это происходит так часто, что используется специальная нотация:

bc =^ hl

где ^ - напоминание о том, что hl повышается одним в качестве побочного эффекта. Правовые вариации:

bc =^ hl, de =^ hl

bc ->^ hl, de ->^ hl

Последние два приводят к тому, что регистры слов хранятся с использованием hl в качестве указателя памяти.

Наконец, часто необходимая последовательность:

a = \*hl

привет + 1

h = \*hl

l = a

который заменяет hl словом, на которое он указывает , используя a, может быть записан:

hl =a^ hl

Опять же, a и ^ предупреждают, что a забивается, а hl увеличивается.

**ОШИБКИ**

Единственный способ добавить один и установить флаг переноса-это хитрость:

а + 0401

**ИМЯ**

Управление - прыжки и управление технологическим процессом

**ФУНКЦИЯ**

Управление принимает две формы В компьютере, направляя последовательность операций в рамках данного процесса и направляя взаимодействие между процессами и их окружением. Первый осуществляется условными скачками, вызовами подпрограмм и возвратами; последний обычно реализуется как мешанина особых инструкций, в обоих случаях 8080 является наиболее типичным.

безусловная передача управления называется скачком, который можно записать в виде унарного оператора:

jmp X

который перемещает адрес X в pc (переходит к X) или в виде двоичных операторов

a = 0 jmp M

Последняя последовательность очищает a, а затем переходит к H.

Существует также восемь условных прыжков, которые совершаются только в том случае, если их получает соответствующее государство флага. Есть:

|  |  |
| --- | --- |
| ОПЕРАТОР | ПРЫГАЕТ, ЕСЛИ |
| jnz X | Z == 0 |
| jz X | Z == 1 |
| jnc X | C == 0 |
| jc X | C == 1 |
| jpo X | P == 0 |
| jpe X | P == 1 |
| jp X | S = = 0 |
| jm X | S == 0 |

Чтобы перейти к подпрограмме, команда вызова

вызов X

толкает ПК в стек, затем переходит к X. Это позволяет вернуться с

замачивать

чтобы вернуться к последовательности вызовов, вытащите ПК из стека. Вызовы, как и прыжки, рассматриваются как операторы, а возвраты-как числа.

8000 также имеет полный набор условных вызовов и возвратов, хотя они редко используются:

cnz X rnz

cz X rz

cnc X rnc

cc X rc

cpo X rpo

cpe X rpe

cp X rp

см Х рм

Наконец, можно перейти к вычисленному адресу с помощью

jmp \*hl

который копирует hl в ПК.

Более специализированный способ вызова функции - это команда перезапуска:

rst0; rst1; -..; rst7

Общая форма rstn толкает пк в стек, а затем переходит к абсолютному местоположению 8\*n, то есть к местоположению 0, 8, 16, 24и т. Д. Перезапуски обычно зарезервированы для связи с системными функциями. На самом деле, когда происходит прерывание, 8080 обычно ведет себя так, как если rstnбы в кодовую последовательность была введена команда rstn, причем n поставляется прерывающим оборудованием.

Прерывания могут быть включены (разрешены) путем выполнения

ei

и отключен с помощью

di

если прерывания были отключены и обнаружен el, то Инструкция, следующая за ei, выполняется до того, как прерывания фактически включены.

Op до 256 логических портов может быть адресован 8080 для выполнения байтового ввода/вывода за один раз. Нет никакого требования, чтобы порт действительно присутствовал или чтобы выходной порт был каким-либо образом связан с входным портом, имеющим тот же номер.

Для чтения данных:

in; n

копирует любой входной порт n, посылаемый в регистр a, в то время как

выход; n

представляет содержимое a и адрес порта n внешнему миру.

**ИМЯ**

Техника - 8080 для взрослых

**ФУНКЦИЯ**

Простого знания набора команд Intel 8080 недостаточно, чтобы правильно его запрограммировать, потому что а) набор примитивен, б) он неполон и, следовательно, в) самые банальные операции часто нетривиальны и неочевидны. В этом разделе описывается ряд методов кодирования в зрелом возрасте в такой незрелой среде.

За редким исключением байтовые (или любые другие) операции требуют использования а в качестве аккумулятора; следовательно, аф следует рассматривать как чрезвычайно изменчивую. Например , возврат байтовых значений в a часто неоптимален, потому что a часто требуется только для того, чтобы вернуться из функции! Гораздо важнее знать:

1) Что можно загрузить в a - {b, c, d, e, h, l, \*hl, #, \*bc, \*de, M}

2) Что можно добавить к a - {b, c, d, e, h, l, \*hl, #)

где # -любая однобайтовая константа, а M-ссылка на память. К счастью, все, что может быть загружено в a, также может быть сохранено из a, за исключением, конечно, константы; и то, что может быть добавлено к a, также может быть вычитано, и т. Д.

Таким образом, наиболее общим методом выполнения операции <op> между байтами является:

1) Настройка адреса назначения в bc

2) Настройка адреса источника в hl

3) Напишите "a = \*bc <op> \*hl -<op>> \*bc<op>>".

Естественно, шаги (1) или (2) могут быть опущены, если операнд уже находится в пределах досягаемости.

Чтобы получить операнды в пределах досягаемости, требуется адресная арифметика, которая обычно включает hi в качестве накопителя указателей. Если операнд находится в стеке,

hl = D + sp

позволяет данным при смещении D от sp (также записанным D(sp)) bcбыть адресованными как \*hl. Язык C использует de в качестве "указателя кадра", чтобы не приходилось внимательно следить за состоянием sp. Таким образом, аргументы и автоматические переменные адресуются как D(de), где D работает от 4 вверх для аргументов и от -7 вниз для авто. Последовательность

bc = (hl = 4 + de)

ясно, что нагрузки будут с адресом первого аргумента.

Часто необходимо проследить цепочку указателей, чтобы добраться до операнда. Это известно по-разному как косвенная адресация или разыменование,

Для этого два сложных хода "=^" и "=а^" часто окупаются. Если, например, первый аргумент является указателем на интересующий операнд:

bc =^ (hl = 4 + de) / операнд теперь в \*bc

или:

hl =a^ (hl = 4 + de) / операнд теперь в \*hl

Дальнейшая модификация, такая как добавление постоянного смещения в массив или структуру, обычно требует как bc, так и hl:

hl = 6 + bc -> bc / для bc + 6

или:

hl + (bc = 4) / для hl + 4

Однако если добавляемое смещение меньше показанного, то запись будет короче и быстрее:

британская колумбия + 1 + 1 + 1 / для bc + 3

или:

хл + 1 + 1 + 1 / для hl + 3

и никакие другие регистры не нужны.

Тестирование неизменно требует установки соответствующего флага в f. Для знаковой арифметики существует шесть возможностей:

**ПЕРЕЙТИ ЕСЛИ КОД**

а

a >= b a :: b jp X

a == b a :: b jz X

a != b a :: b jnz X

a > b b :: a jm %

а

Обратите внимание, что все, кроме двух сравнений равенства, дают неправильные результаты, если a и b отличаются на 128 или более. Этого трудно избежать, если только тест не может быть преобразован в сравнение без знака, показанное ниже.

Чтобы сравнить байт в b с нулями

**ПЕРЕЙТИ ЕСЛИ КОД**

b

b >= 0 a = b | a jp X

b == 0 a = b | a jz X

b != 0 a = b | a jnz X

b > 0 a - a :: b jm X

b

Неподписанные тесты такие же, как и подписанные, за исключением того, что вместо них проверяется флаг переноса:

**ПЕРЕЙТИ ЕСЛИ КОД**

а

a >= b a :: b jnc X

a == b a :: b jz X

a != b a :: b jnz X

a > b b :: a jc X

а

Эти тесты верны для всех значений a и b. Значимые нулевые сравнения для беззнаковых коллапсов к равенству и неравенству, которые такие же, как и выше:

**ПЕРЕЙТИ ЕСЛИ КОД**

b == 0 a = b | a jz X

b != 0 a = b | a jnz X

Арифметика с множественной точностью, которая часто включает в себя даже двухбайтовые вычисления, требует разумного шага регистров указателей и пристального внимания к флагу переноса. К счастью, конструкторы 8080-х позаботились о том, чтобы эти операции редко сталкивались. Прототипом последовательности множественной точности является целочисленное вычитание, представленное здесь для \*bc - \*hl:

a = \*be - \*hl -> \*bc = \*(bc+1) -^ \*(hl+1) ->> \*bc

Эта цепочка может быть расширена до произвольной длины или помещена в цикл; сделать восемь байт, например:

de => sp

e = 7

a = \*bc + \*hl -> \*bc

L:

a = \*(bc + 1) -^ \*(hl + 1) -> \*bc

e - 1 jnz L

де

Очевидно, что сложение может быть выполнено таким же образом с помощью оператора"+^".

Сравнение нескольких прецизионных операндов аналогично однобайтовым тестам:

**ПЕРЕЙТИ ЕСЛИ КОД**

\*bc < \*hl a = \*bc- \*hl = \*(bc+1) -^ \*(hl+1) jm X

\*bc >= \*hl a = \*bc - \*hl = \*(bc+1) -" \*(hl+1) jp X

\*bc == \*hl a = \*bc :: \*hl jnz T = \*(bc+1) :: \*(hl+1) jz X; T:

\*bc != \*hl a = \*bc :: \*hl jnz X = \*(bc+1) :: \*(hl+1) Jnz X

\*bc > hl> sp <= bc <= hl => bc =<= hl =>> hl; / see (\*<= hl =>>bc)

\*bc <= \*hl sp <= bc <= hl => bc => hl; / see (»bo >= \*hl)

Опять же, они неверны, если аргументы отличаются достаточно широко, чтобы вызвать переполнение (jp/jm), но всегда верны для сравнения без знака (jnc/jc).

Чтобы проверить целое число на \*hl против нуля:

**ПЕРЕЙТИ ЕСЛИ КОД**

\*hl

\*hl >= 0 a = \*(hl + 1) | a jp X

\*hl == 0 a = \*hl | \*(hl + 1) jz X

\*hl != 0 a = \*hl | \*(hl + 1) jnz X

\*hl > 0 a - a - \*(hl + 1) jm X

\*hl

Все эти операции легко распространяются на большее количество байтов точности. Обратите внимание, однако, что C представляет лонги и удваивается таким образом , что часто только \*(hl + 1), или \*hl и \*(hl + 1) не более, необходимо проверить, чтобы определить значение целого.

В качестве заключительного замечания следует отметить, что пространство в стеке часто должно покупаться и продаваться по мере ввода и удаления контрольных блоков, в общем случае

hl = N + sp -> sp

это лучший способ добавить константу N в sp. Если N четное и меньше 10, однако, это быстрее, чтобы нажать или поп aрегистр несколько раз:

sp <= af

sp => af =>> af / pop 4 байта

af является хорошим кандидатом на выброс мусора, так как он очень волатильен.

**ИМЯ**

Резюме - А-Естественное вкратце

**ФУНКЦИЯ**

Символы могут быть определены как один из следующих:

1. однобайтовые константы (точность с точностью до двух байт)
2. двухбайтовые константы, которые могут быть перемещаемыми адресами
3. ссылки на память

Определение происходит либо с помощью:

1. X := Y определяет X как Y
2. X: определяет X как .

Генерация кода направлена;

1. в текстовый раздел изначально, для литералов в конце и после ". := .text".
2. в раздел данных, после ". := .data"

Генерация кода происходит для:

a) однобайтовый термин, дающий байт этого значения

b) только двухбайтовый термин, дающий два байта этого значения

в) строка, дающая байт для каждого символа в строке

d) резервирование apace, дающее n нулевых байтов для ". := .[n]"

д) двоичный оператор X op Y, дающий соответствующие байты команд

Условия изменяются следующим образом:

1. унарный оператор вида <op><op> X
2. индексы вида X[n], где n - абсолютная константа.

Строка-это либо литеральная строка, либо результат оператора конкатенации запятых. Допустимыми операндами для запятой являются однобайтовые константы, двухбайтовые абсолютные константы и строки.

Унарные операторы это:

1. "&" , чтобы сделать двухбайтовую константу
2. "\*" для создания ссылки на память или косвенной записи в регистр
3. "+" ничего не делает с абсолютным числом
4. "-" чтобы отрицать абсолютное число
5. "!- к единицам добавляется абсолютное число
6. "=" для генерации двухбайтового литерала
7. public чтобы сделать символ известным во всех файлах

Двоичные операторы и их допустимые операнды используют следующие сокращения:

1. Cr - это сокращение от {a, b, c, d, e, h, l, \*hl}.
2. Ir - это сокращение от {bc, de, hl, sp}.
3. N-это однобайтовая константа.
4. NN-это двухбайтовая константа.
5. М-это ссылка на память.
6. X = Y эквивалентно Y ->> X.
7. X <= Y эквивалентно Y => > X.
8. X <><> Y эквивалентно Y <><> X.

Операции оцениваются слева направо, за исключением тех случаев, когда они изменены круглыми скобками. Левый операнд вычисляется первым и сохраняется для всех последующих операций. Допустимыми формами являются:

|  |  |  |  |
| --- | --- | --- | --- |
| **Байтовые операции** | | **слову операции** | |
|  | а = \*до н. э. | | ИК = П |
|  | а -> \*БК | | ИК = НН |
|  | = а \*де | | гл = м |
|  | в -> \*де | | гл -> м |
|  | а = м | | СП = гл |
|  | а -> м | | вс = де |
|  | ЧР = п | | до н. э. = гл |
|  | ЧР = ЧР | | де = ВС |
|  | ЧР + 1 | | де = гл |
|  | а + КР | | гл = БЦ |
|  | а + н | | гл = де - |
|  | КР - 1 | | БК => п |
|  | а - СГ | | де => П |
|  | А - Н | | гл => П |
|  | А +^ КР | | АФ => П |
|  | А +^ Н | | БК <= СП |
|  | а -^ СГ | | де <= СП |
|  | а -^ п | | гл <= СП |
|  | и СГ | | АФ <= СП |
|  | А Н | | гл <> \*СП |

|  |  |
| --- | --- |
| а ^ пр | гл <> де |
| В ^ Н | гл + ИК |
| а | КР | ИК + 1 |
| В | П В | ИК – 1 |
| а :: ЧР | г. до н. э. =^ гл |
| а :: н | де =^ гл |
| а <^> л | до н. э. ->^ гл |
| а <\*> -1 | - де ->^ гл |
| а <\*> 1 | гл =а^ гл |
| а <+> -1 |  |
| а <+> 1 |  |
| в <+> -1 |  |
| а <-> 1 |  |
| а <-> +1 |  |
| а =! а |  |

**УСЛОВНЫЕ ОБОЗНАЧЕНИЯ**

jmp X call X ret

jnz X cnz X rnz

jz X cz X rz

jnc X cnc X rnc

jc X cc X rc

jpo X cpo X rpo

jpc X cpc X rpc

jp X cp X rp

jm X cm X rm

**ДРУГИЕ ПРЕДОПРЕДЕЛЕННЫЕ КОНСТАНТЫ**

cmo rst0

daa rst1

di rst2

ei rst3

hlt rst4

в rst5

nop rst6

выход первым 7

stc