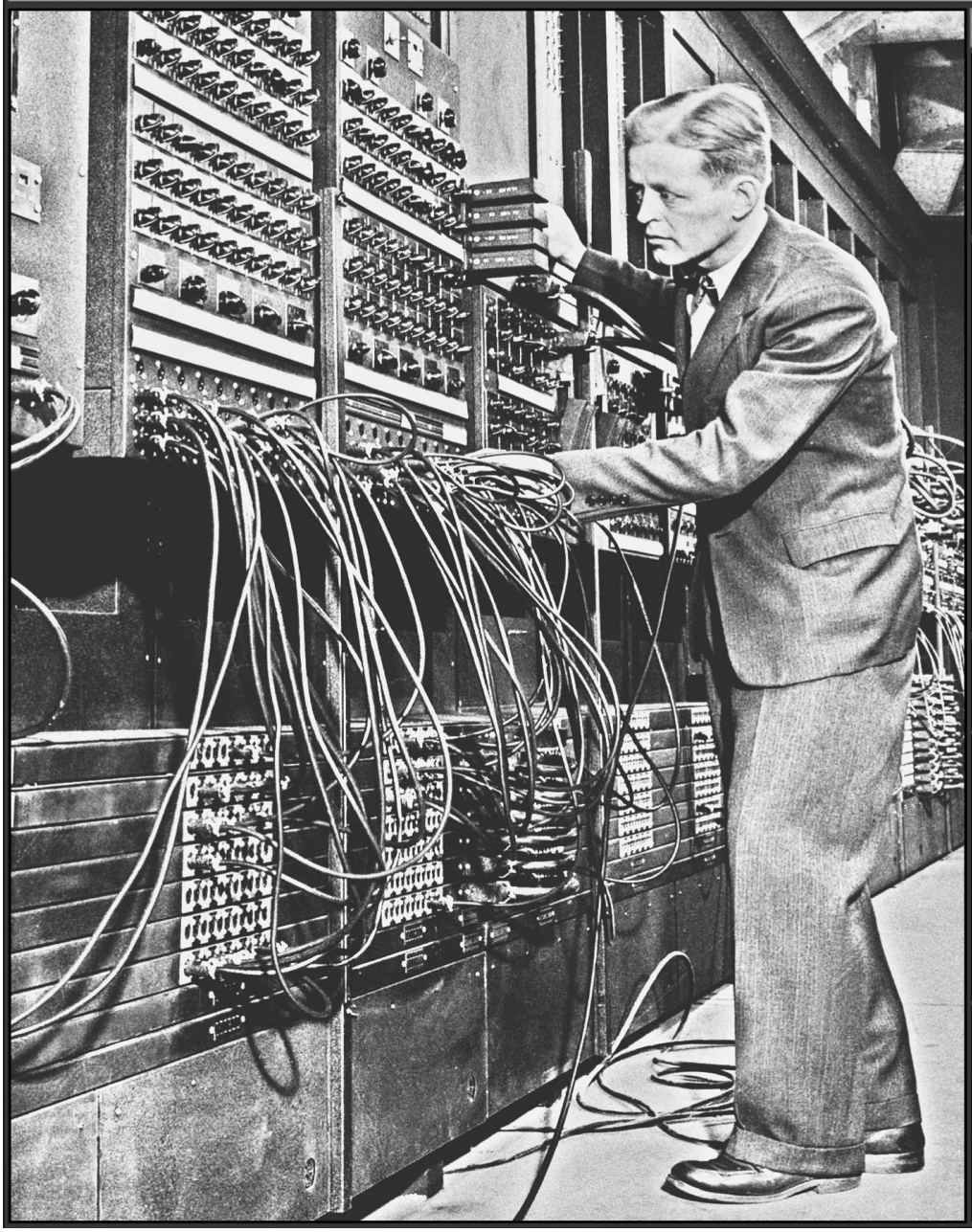
NedoPC is not PC

NedoPC

четвертый номер 2006



анонс

Содержание номера

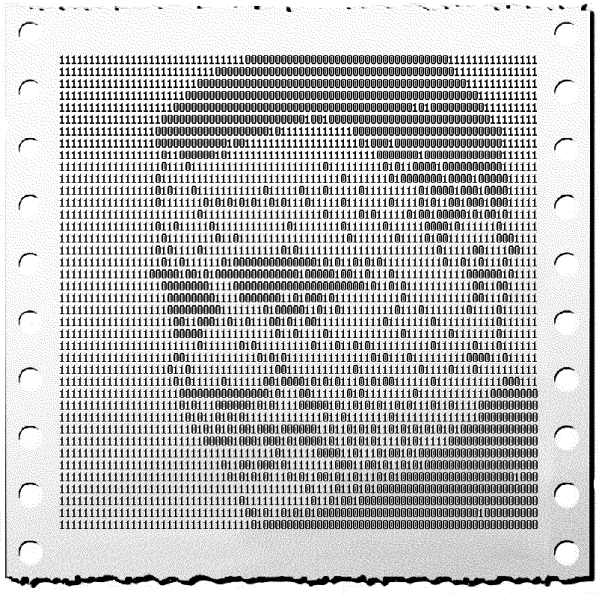
Номер скомпилирован из TeX исходников November 7, 2006. Вёрстка номера осуществлялась в операционной системе Slackware Linux с помощью системы подготовки текстов LaTeX, графических редакторов GIMP и Xfig. С 2006 года издание распространяется бесплатно в бумажном (A5) и электронном (PDF) виде. Обо всех нарушениях условия бесплатного распространения просьба сообщать в виртуальное представительство редакции журнала:

E-mail: journal@nedopc.org

http://journal.nedopc.org

http://www.nedopc.org

Колонка редактора



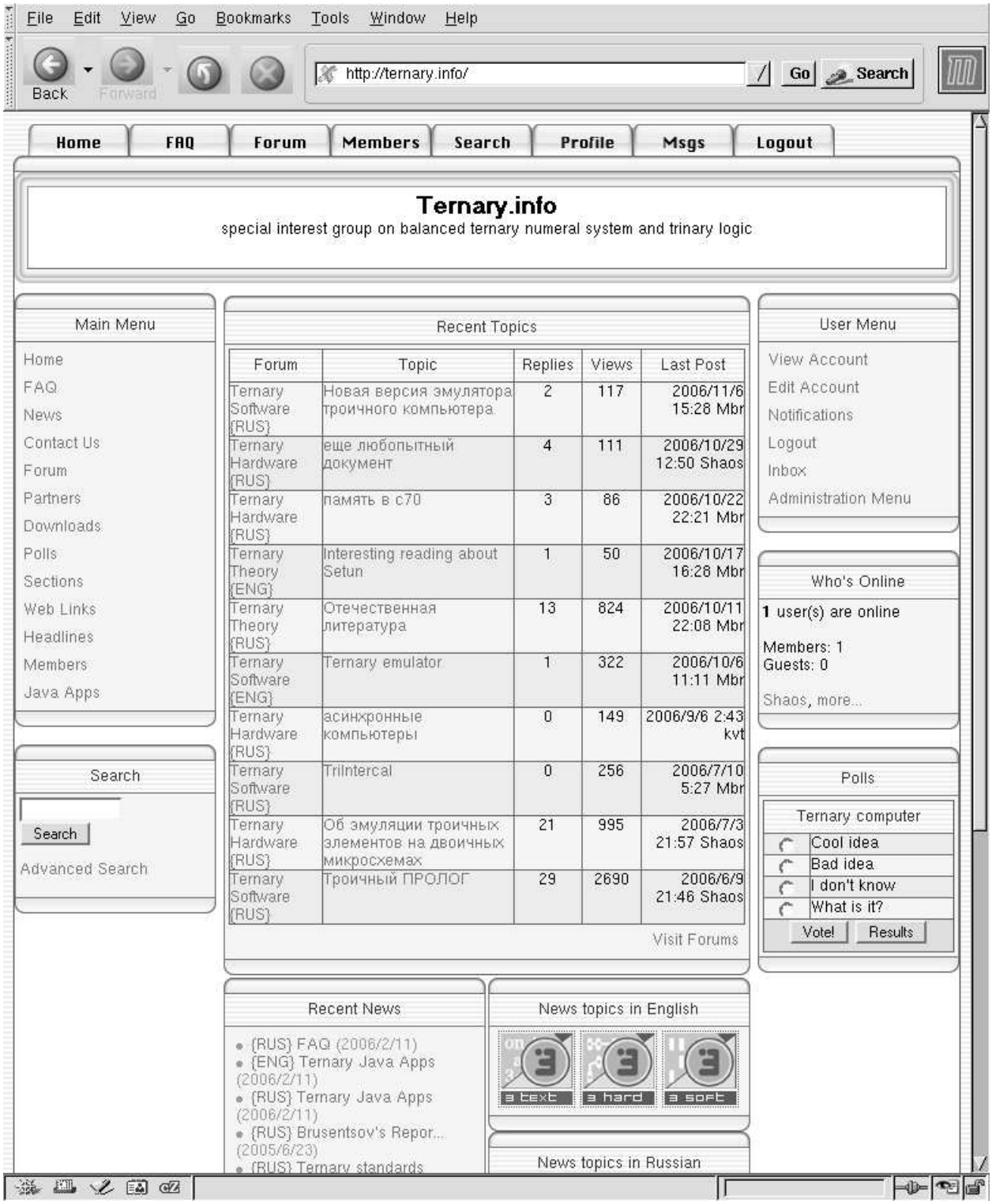
Мы собирались с силами слишком долго и, наконец собравшись, выпустили четвёртый номер нашего издания! Надеемся не слишком поздно и ты, уважаемый читатель, не успел о нас забыть. За это время мы успели полностью встать на рельсы открытых исходников и наш номер целиком свёрстан в свободно распространяемой системе LaTeX (за исключением некоторого графического материала, предоставленного нам со стороны) под управлением свободно распространяемой операционной системы Linux (а точнее Slackware 10.0). Кроме LaTeXa использовались программы GIMP и Xfig. Надеюсь издание от этого ничего не потеряло, а только приобрело, в том числе и более классический типографский стиль.

Система LaTeX предоставляет прекрасные возможности по вводу математических формул на собственном языке их описаний, а также даёт всю необходимую полиграфическую функциональность, в том числе и конвертор в PDF формат, в котором мы планируем распространять наше издание в скором будущем с нашего сайта http://www.nedopc.org. Масса времени была потрачена на ручное преобразование текстов и иллюстраций для LaTeXa (именно поэтому номер не был закончен к началу фестиваля Chaos Construction 2006, как планировалось — теперь пытаемся уложиться к началу KidSoft'2006 ;). В будущем хотелось бы переложить работу по подготовке материала на самих авторов — ведь в языке LaTeX нет ничего сложного (многие типографии мира принимают в печать научные статьи, подготовленные в LaTeX).

А теперь перейдём к освещению содержания номера — существенная часть страниц поглощена материалом одного из наших читателей (точнее можно сказать — одного из наших авторов), который рассказывает о самостоятельном создании внешней звуковой карты для PC на базе музыкального чипа YM2612 (думается что сей девайс возможно подключить не только к PC, т.к. управляется устройство через LPT-порт). Также в номере можно найти новые материалы о троичной системе счисления (ещё один новый автор!) — введение в троичную арифметику и использование алгоритма Хаффмана для кодирования данных троичными кодами. Ещё у нас есть статья о вычислениях с произвольной точностью на троичном процессоре, а также материал, посвящённый одной из самых необычных систем команд процессора.

После прочтения номера не забудь заглянуть на наш форум http://forum.nedopc.org для общения по тематике наших статей и не только!

Главный редактор Shaos



http://ternary.info

Троичная арифметика

Автор: Александр Никитич[1]

Рассматриваются троичная симметричная система счисления (ТСС) и операции производимые в ней. Благодаря тому, что основание нечётно (основание системы 3), в такой системе возможно симметричное относительно нуля представление значений: отрицательное, нулевое, положительное. Такую систему ещё называют троичной симметричной (уравновешенной) системой счисления.

В троичной системе веса соседних разрядов различаются в три раза. В общем виде число в троичной системе можно представить как сумму целых степеней числа 3, причём знак слагаемого определяется значением соответствующего разряда. Иными словами: $a_n \cdot 3^n + a_{n-1} \cdot 3^{n-1} + \ldots + a_2 \cdot 3^2 + a_1 \cdot 3 + a_0 + a_{-1} \cdot 3^{-1} + a_{-2} \cdot 3^{-2} + \ldots + a_{-m+1} \cdot 3^{-m+1} + a_{-m} \cdot 3^{-m}$, где $a_i \in [-1, 0, 1], n, m, i \in N$. Причём $a_n \cdot 3^n + a_{n-1} \cdot 3^{n-1} + \ldots + a_2 \cdot 3^2 + a_1 \cdot 3 + a_0$ — целая часть числа, а $a_{-1} \cdot 3^{-1} + a_{-2} \cdot 3^{-2} + \ldots + a_{-m+1} \cdot 3^{-m+1} + a_{-m} \cdot 3^{-m}$ — дробная часть.

Существует несколько вариантов обозначения алфавита троичной системы счисления:

- Н.П. Брусенцов [1] ввёл для обозначения символы $\bar{1}$, 0, 1; (для обозначения $\bar{1}$ может использоваться символ i).
- На сайте Ternary.info [2] используются символы *N, 0, P*;
- Автор же использует символы $-, 0, +$.

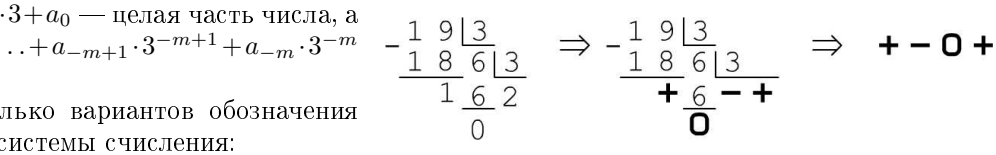
Десятичное представление троичного числа выражается как сумма произведений значения разряда на соответствующую этому разряду степень числа 3 (в десятичном представлении).

Так в троично-симметричной системе число $+-+$ в десятичном представлении является числом 7: $+-+_3 = 1 \cdot 3^2 + (-1) \cdot 3^1 + 1 \cdot 3^0 = 7$.

[1] E-mail автора: neutec@yandex.ru

Для перевода из десятичной системы в троичную, можно воспользоваться следующим алгоритмом:

1. выполняем деление в десятичной системе исходное число на 3;
2. если остаток от деления равен 2, записываем его как $-$, а к результату от деления добавляем 1;
3. если результат меньше 3, начинаем записывать результат перевода, п.5;
4. делим результат на 3, затем повторяем действия описанные в п.2;
5. если результат 2, то переписываем его как $-+$;
6. переписываем полученные значения остатков (снизу-вверх), начиная с последнего результата деления.



Данный способ больше пригоден для преобразования с вычислительным устройством, так как если необходимо преобразовать большое число, придется выполнять соответственно большое число операций деления.

Ключевая особенность ТСС — наличие знака числа в самом алфавите, т.е. однозначно определяется знак числа по самому числу. Если ведущий ненулевой разряд отрицателен, то и само число является отрицательным. Изменение знака числа производится инвертированием каждого разряда числа: положительный разряд меняется на отрицательный и наоборот, ноль остается без изменений:

$$5_{10} = +--$$

$$-5_{10} = -++$$

Другой важной особенностью является механизм округления чисел — простым отбрасыванием младших разрядов получается наилучшее при данном оставшемся количестве цифр приближение этого числа и округление не требуется. Это следствие того, что абсолютная величина части числа, представленной отбрасываемыми младшими цифрами, никогда не превосходит половины абсолютной величины части числа, соответствующей младшей значащей цифре младшего из сохраняемых разрядов.

Округлим число $0,+++{+}+ = 0,493827$ до 3 знаков после запятой: $0,+++ = 0,48148$. Округление до 2 знаков даёт: $0,{+}+ = 0,4444(4)$. Округление до 1 знака — $0,+ = 0,3333(3)$.

Сложение

Сложение производится по общим правилам для позиционных систем с учетом следующей таблицы:

слагаемое	+	+	+	-	-
слагаемое	+	0	-	0	-
сумма	-	+	0	-	+
перенос	+	0	0	0	-

Оптимально в троичных ЭВМ производить сложение сразу 4 слагаемых, одно из которых является переносом от предыдущей операции. Такая постановка обусловлена тем, что только при 5 слагаемых может появиться второй знак переноса, в двоичных системах так можно складывать только 2 слагаемых.

$$\begin{array}{l} + \\ + \\ + \\ + \\ \hline ? \end{array} \quad \Rightarrow \quad \begin{array}{l} \\ +- \\ +- \\ \hline ++ \end{array}$$

Вычитание

Операция вычитания выполняется через сложение с инвертированием одного из слагаемых, т.е одно из слагаемых меняет знак:

$$(+0-) - (-+) = (+0-) + (+-) = (+0+)$$

Умножение

Операция умножения производится по общим правилам для позиционных систем с учетом следующей таблицы:

*	+	0	-
+	+	0	-
0	0	0	0
-	-	0	+

Умножение 8 на 8, в результате получаем 64:

$$\begin{array}{l} \times\ +\,0\,- \\ \quad +\,0\,- \\ \hline \quad -\,0\,+ \\ +\,0\,0\,0 \\ +\,+\,0\,- \\ \hline +\,-\,+\,0\,+ \end{array}$$

Умножение в ТСС на 3^n, где $n > 0, n \in N$, сводится к добавлению к числу n младших разрядов с нулевым значением:

$$4 = ++;$$

$$12 = ++0;$$

$$36 = ++00.$$

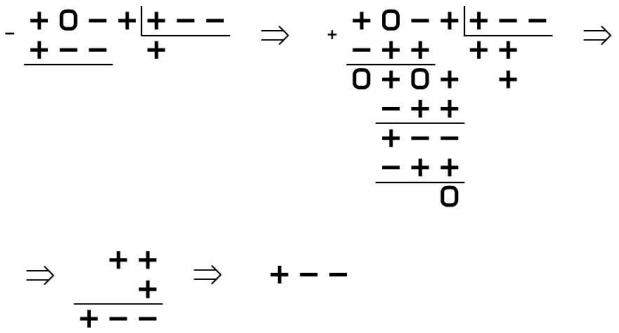
Также эта операция называется троичный сдвиг влево.

Деление

Операция деления производится по общим правилам для позиционных систем. При делении столбиком, если разрядность остатка от деления не уменьшилась, необходимо произвести деление без добавления следующего разряда делимого, а результат записать под предыдущем результатом, затем необходимо сложить полученные результаты поразрядно. Удобно заменять операцию вычитания — операцией сложения, для этого вычитаемое необходимо

инвертировать, после чего можно производить сложение.

Деление 25 на 5:



Операцию вычитания заменяем сложением, в результате второй операции получается остаток с разрядностью равной разрядности делителя, т.е. необходимо произвести деление остатка без займа следующего разряда делимого. Полученный результат записываем под предыдущим результатом, а затем производим поразрядное сложение полученных результатов.

Аналогично операции умножения при делении числа на 3^n, где $n > 0, n \in N$, происходит троичный сдвиг вправо:

$$18 = +-00;$$

$$6 = +-0;$$

$$2 = +-;$$

$$\frac{2}{3} = +, -.$$

Признак делимости на 2

Целое число a в ТСС можно представить в виде $a = \overline{a_n a_{n-1} \dots a_1 a_0} = 3^n \cdot a_n + 3^{n-1} \cdot a_{n-1} + \dots + 3^1 \cdot a_1 + 3^0 \cdot a_0 = (3^n - 1) \cdot a_n + (3^{n-1} - 1) \cdot a_{n-1} + \dots + (3^2 - 1) \cdot a_2 + (3 - 1) \cdot a_1 + a_n + a_{n-1} + \dots + a_1 + a_0$, где $a_i \in \{-1, 0, 1\}$, $i = 0, 1, \dots, n$, $a_n \neq 0$. Обозначим $b = (3^n - 1) \cdot a_n + (3^{n-1} - 1) \cdot a_{n-1} + \dots + (3 - 1) \cdot a$, $c = a_n + a_{n-1} + \dots + a_1 + a_0$, т.е. $a = b + c$.

Рассмотрим число b — множителями a_i являются $(3^k - 1)$. При любом $k \geq 1$ $(3^k - 1) = (3 - 1) \cdot (3^{k-1} + 3^{k-2} + \dots + 3 + 1) = 2 \cdot (3^{k-1} + 3^{k-2} + \dots + 3 + 1)$, следовательно, можно записать $b = 2 \cdot (\dots)$, т.е. b делится на 2 без остатка.

Следовательно, число $a = b + c$ делится на 2 тогда, когда делится на 2 число $c = a_n + a_{n-1} + \dots + a_1 + a_0$, которое представляет собой сумму цифр числа a, т.е. число в ТСС делится на 2 без остатка (т.е. число — чётное), если сумма его цифр делится на 2.

Числа, которые делятся на 2			Числа, которые не делятся на 2		
ТСС	Сумма цифр	ДПСИ	ТСС	Сумма цифр	ДПСИ
+-	0	2	+	1	1
++	2	4	+0	1	3
+-0	2	6	+–	-1	5
+0-	2	8	+-+	1	7
+0+	2	10	+00	1	9
++0	2	12	++-	1	11
+—	4	14	+++	3	13

Признак делимости на 3

Число a в ТСС можно представить в виде $a = \overline{a_n a_{n-1} \dots a_1 a_0} = 3^n \cdot a_n + 3^{n-1} \cdot a_{n-1} + \dots + 3^1 \cdot a_1 + 3^0 \cdot a_0$, где $a_i \in \{-1, 0, 1\}$, $i = 0, 1, \dots, n$, $a_n \neq 0$. Зададим число $b = a - a_0$, т.е. $b = 3^n \cdot a_n + 3^{n-1} \cdot a_{n-1} + \dots + 3^1 \cdot a_1 = 3 \cdot (3^{n-1} \cdot a_n + 3^{n-2} \cdot a_{n-1} + \dots + 3 \cdot a_2 + a_1)$. Отсюда следует, что число b делится на 3. Число $a = b + a_0$ делится на 3 тогда и только тогда, когда a_0 делится на 3, этому соответствует значение $a_0 = 0$, таким образом, числа в ТСС делятся на 3, если младший разряд равен нулю.

Числа, которые делятся на 3		Числа, которые не делятся на 3	
ТСС	ДПС	ТСС	ДПС
+0	3	+	1
+-0	6	+-	2
+00	9	++	4
++0	12	+0-	8

Литература

1. http://cs.msu.su/jetspeed/portal/content/view/person/general?id=4053751
2. http://www.ternary.info/
3. http://flags.pfu.edu.ru/content/Lib/math/Gromov/OralExams/chapt1.pdf

А тут могла бы быть ваша реклама

journal@nedopc.org

Вычисления произвольной точности на троичном процессоре

Автор: Mac Buster[2]

Однажды в форуме NedoPC Александром Шабаршиным и Виктором Рощупко была высказана одна весьма примечательная идея, заключающаяся в создании одноразрядного троичного (однотритного) вычислителя. В то время она была обойдена вниманием и скорее всего рассматривалась нами как представляющая исключительно теоретический интерес, поскольку тогда мы только-только начинали свои исследования в области уравновешенной троичной системы счисления, а так же в связи с тем, что ещё не существовало никакой троичной элементной базы.

Теперь, когда появилось некоторое, пусть и небольшое количество троичных элементов, можно попробовать реализовать эту идею в виде несложного периферийного устройства, предназначенного, скажем, для проверки эффективности реализации алгоритмов работы с данными, представленными в уравновешенном троичном коде, и эффективности троичной логики. На мой взгляд, можно создать такое устройство двумя способами: разработать полноценный троичный вычислитель, или, используя двоично-троичное кодирование, построить своеобразный периферийный троичный вычислитель на основе недорогой микроконтроллерной отладочной платы. Если первый вариант потребует от разработчиков немало сил и времени, то для воплощения в жизнь второго варианта надо не так уж много времени и для него вполне может подойти даже простейшая отладочная плата на основе какого-либо микроконтроллера начального уровня (например, PIC или AVR), оснащённая по крайней мере одним портом RS-232 или USB и небольшим объемом оперативной памяти.

[2] E-mail автора: mbr@ternary.info

Одним из достоинств реализации именно второго варианта является возможность загрузки программы в память такой платы и выполнение вычислений после отключения устройства от компьютера, который использовался для написания программы и её передачи в устройство для выполнения.

Для чего нам может пригодиться такой откровенно минималистический периферийный вычислитель? С его помощью можно будет не только проверить насколько хорошо вам удалось разобраться с выполнением элементарных арифметических операций, производимых над числами представленными в уравновешенной троичной форме, или провести исследование эффективности какого-либо специфического троичного алгоритма, но и построить весьма интересную систему - троичный математический процессор для вычислений с произвольной точностью. Устройство может быть организовано таким образом, что будет производить каждое арифметическое действие ровно с той точность, которая требуется разработавшему программу программисту. При этом не надо будет опасаться выхода из диапазона представления разрядной сетки и принимать соответствующие меры предосторожности, как это происходит при использовании двоичных микроконтроллеров и микропроцессоров традиционной архитектуры.

При реализации такой системы потребуется разделить доступную оперативную память на три независимых непересекающихся области: *стек, программа, данные.* Область данных будет разбита на две части: *область описателей* и *область собственно данных.* Описатель в свою очередь состоит из *смещения* S и *длины* L. Смещение S - это указатель адреса, начиная с которого идут данные длиной в L трит (элементарных троичных разрядов).

Система команд предполагается минимальной. Команды выполняющие арифметические операции будут содержать указание о требуемой точности результата вычисления в тритах.

Процессор предоставляет программисту:

- три 9-разрядных регистра общего назначения: X, Y, Z;
- регистр-счётчик команд - PC, содержащий адрес текущей выполняемой команды;
- регистр-указатель стека - SP;
- регистр состояния процессора - CF (содержит набор флагов).

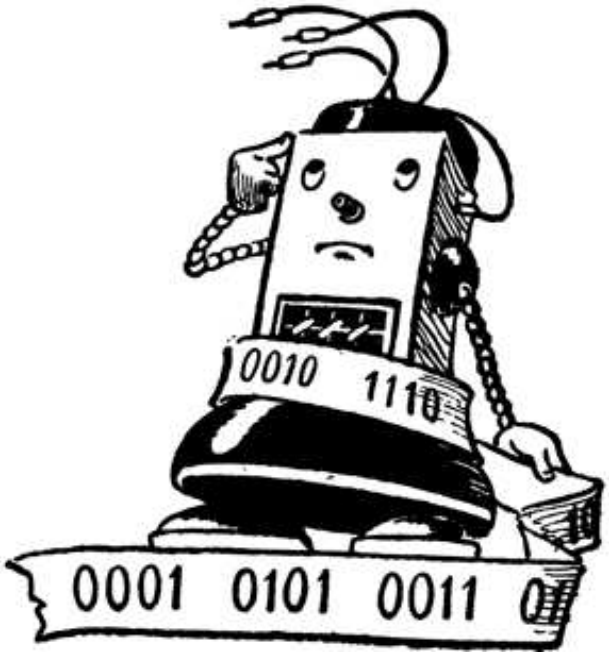
Примерный перечень команд процессора с разделением по функциональным группам:

1. Пересылка данных:
   - загрузка из памяти в регистр;
   - запись из регистра в память;
   - пересылка из регистра в регистр.
2. Арифметика:
   - сложение с переносом;
   - вычитание с займом;
   - умножение с переносом;
   - деление;
   - сравнение двух чисел;
   - определение знака числа;
   - определение чётности числа;
   - выделение целой части числа;
   - выделение дробной части числа;
   - смена знака числа;
   - округление числа.
3. Логика:
   - max;
   - min;
   - mean;
   - shift up;
   - shift down.
4. Управление выполнением программы:
   - условный переход;
   - безусловный переход;
   - возврат.
5. Управление процессором:
   - останов;
   - выдача значения регистра;
   - запрет и разрешение прерываний.

Перечисленных команд должно хватить для реализации значительной части алгоритмов обработки данных.

Кроме реализации команд потребуется небольшой набор функций для обеспечения обмена данными между платой, на которой выполняется программа-имитатор периферийного троичного процессора, с компьютером, на котором производится разработка программного обеспечения и обработка результатов выполнения этого ПО.

Если кого-нибудь из читателей заинтересовала эта идея, прошу писать мне для обсуждения деталей реализации, или напишите отзыв в наше издание (или на форум http://forum.nedopc.org). Будем рады вашим предложениям и замечаниям.



http://vntb.ru

Троичное дерево Хаффмана

Автор: Александр Никитич[3]

Построение троичных кодовых деревьев по алгоритму Хаффмана. Оценка эффективности по сравнению с двоичными кодовыми деревьями.

Алгоритм Хаффмана служит для получения префиксных кодов — кодов переменной длины, которые позволяют осуществлять экономное кодирование данных. Например, при экономном кодировании символьной информации часто встречаемые символы кодируются короткими кодами, в то время как редко встречающиеся — длинными. О том, что такое префиксные коды и алгоритмы их получения, описано в [1].

Рис.1 Построение системы двоичных префиксных кодов.

Для начала рассмотрим пример построения довичных кодовых деревьев по алгоритму Хаффмана. В качестве кодируемого сообщения выберем слово "ГОЛОГРАММА". Статистика появления символов в этом сообщении: Г(2), О(2), Л(1), Р(1), А(2), М(2). Упорядоченный список будет выглядеть так: А(2), Г(2), М(2), О(2), Л(1), Р(1). Построение системы двоичных префиксных кодов для этого сообщения показано на рис.1.

Рассмотрим особенности построения троичных кодовых деревьев по алгоритму Хаффмана. Как и в случае с двоичным вариантом на начальном этапе каждому символу информационного алфавита ставится в соответствие вес, равный частоте (вероятности) появления этого символа в исходных данных. Символы помещаются в список, который сортируется по убыванию весов. На каждом шаге (итерации) три последних элемента списка объединяются в новый элемент, который затем помещается в список вместо трёх объединяемых элементов. Новому элементу списка ставится в соответствие вес, равный сумме весов замещаемых элементов. Каждая итерация заканчивается упорядочиванием полученного нового списка, который всегда содержит на два элемента меньше, чем старый список.

Первая особенность троичного алгоритма заключается в том, что на самом первом шаге объединять нужно либо два, либо три элемента. Дело в том, что если всегда на первом шаге выбирать по три элемента, в некоторых случаях в результирующем дереве на первом уровне может оказаться пустая ветвь, что влечет серьезную потерю в эффективности. Для предотвращения такой ситуации, необходимо учитывать количество элементов исходного списка. В случае, когда количество чётно, на первом шаге необходимо вместо трёх последних элементов, объединять только два. Тем самым пустая ветвь окажется на самом нижнем уровне — эффективность не снизится. В случае нечётного количества — объединяются все три элемента.

Эта особенность вызвана тем, что для более эффективного кодирования необходимо, чтобы первый уровень состоял из трёх элементов, а так как на каждом шаге отнимается по два элемента, то получается, что при таком подходе эффективно кодируется список с нечётным количеством элементов: $n = 3 + 2k$. Поэтому при чётном количестве элементов первоначально объединяются два элемента, т.е. отнимается только один элемент, в результате чего получаем список с нечётным количеством элементов.

[3] E-mail автора: neutec@yandex.ru

Рассмотрим пример построения троичных кодовых деревьев по алгоритму Хаффмана, взяв в качестве кодируемого то же самое сообщение: "ГОЛОГРАММА". Напомним как выглядит упорядоченный список символов для этого сообщения: А(2), Г(2), М(2), О(2), Л(1), Р(1). Список состоит из чётного количества элементов (шесть символов), поэтому на первом шаге объединять следует только два последних элемента вместо трёх. Построение системы троичных префиксных кодов представлено на рис.2.

Рис.2 Построение системы троичных префиксных кодов.

Как видите, построение троичных кодовых деревьев по алгоритму Хаффмана практически не отличается от двоичных, исключение составляет только первый шаг.

В случае двоичного дерева мы закодировали наше сообщение двоичной последовательностью:

$$000101101000011101001001 01$$

В случае же троичного — кодовая последовательность будет такой:

$$+0+00+0++00--0-0--$$

В данном конкретном случае получаем 26 двоичных символов против 18 троичных — троичное сообщение в 1,44 раза компактнее.

Эффективность кодирования в префиксных кодах можно оценить по высоте дерева Хаффмана. Чем оно ниже, тем выше эффективность.

Количество шагов для построения двоичного дерева можно вычислить по формуле:

$$k = n - 1$$

Для построения троичного дерева формула будет выглядеть так:

$$k = \frac{n-1}{2}$$

Следовательно, для построения троичного дерева требуется в два раза меньше шагов (при $n > 2$), где n — количество элементов.

Максимальная высота для n-го количества элементов будет, если на каждом шаге будет объединяться два элемента списка и предыдущие объединение элементов, минимальной — когда на каждом шаге объединяются элементы.

Максимальная высота двоичного дерева:

$$h_{max} = n - 2$$

Максимальная высота троичного дерева:

$$h_{max} = \frac{n-3}{2}$$

Эффективность построения троичного дерева по сравнению с двоичным при максимальной высоте дерева составляет:

$$e_{max} = \frac{2n-4}{n-3} = 2 + \frac{2}{n-3}$$

то есть более чем в 2 раза.

Минимальная высота двоичного дерева:

$$h_{min} = \log_2 n$$

Минимальная высота троичного дерева:

$$h_{min} = \log_3 n$$

Эффективность построения троичного дерева по сравнению с двоичным при минимальной высоте дерева составляет:

$$e_{min} = \frac{\log_2 n}{\log_3 n} = 1,585.$$

Заключение: построение троичного дерева в два раза быстрее (по количеству шагов) и в полтора раза эффективнее (по длине кодов), чем двоичное дерево.

Литература

1. Семенюк В. Экономное кодирование дискретной информации. — СПб., 2001. – 115 с.

2. http://www.compression.ru

Внешняя звуковая плата с FM синтезом звука

Автор: Romanich[4]

В этой статье рассказывается о создании внешней звуковой карты (для краткости далее в тексте ВЗК) для IBM совместимого персонального компьютера (IBM PC).

Звуковая карта собрана на базе музыкального процессора YM2612 (производитель Yamaha) или его аналогов (TA-07, PCS8593). ВЗК управляется компьютером через LPT порт, питание берётся от USB-порта (+5V). ВЗК содержит предварительный усилитель звука, выход которого может подсоединяться к наушникам, активным колонкам или усилителю мощности.

Также в статье рассматривается ряд вопросов, связанных с программированием микросхемы YM2612: общение с регистрами звукового чипа, формат музыкальных файлов *.GYM, рассмотрена возможность воспроизведения 8-битных оцифровок с частотой дискретизации 0...44100 Гц.

Приведены все необходимые сведения для создания, программирования и использования ВЗК. В конце статьи даны необходимые ссылки на программы и документацию.

Всё что рассказано в данной статье, сделано, проверено и работает под управлением операционной системы PC-DOS v7.0 и Windows-98.

Принципиальная схема

Схема ВЗК представлена на соседней странице. Основа схемы — звуковой чип (музыкальный процессор, микросхема ...— кому как больше нравится) YM2612 от японской фирмы Yamaha. Данная микросхема позволяет получить множество звуковых эффектов на основе FM-синтеза. Тип частотного синтеза — OPN2. Это значит, что каждый инструмент имеет 4 оператора, которые соединяются по типу N. Учитывая то, что это — крайне редкая микросхема, довожу до сведения, что вместо неё можно поставить аналоги: TA-07 или PCS8593, выпаянные из неисправных плат игровой приставки SEGA (MegaDrive a.k.a. Genesis). То есть наша ВЗК будет автоматически способна проигрывать музыкальные композиции от игр SEGA!

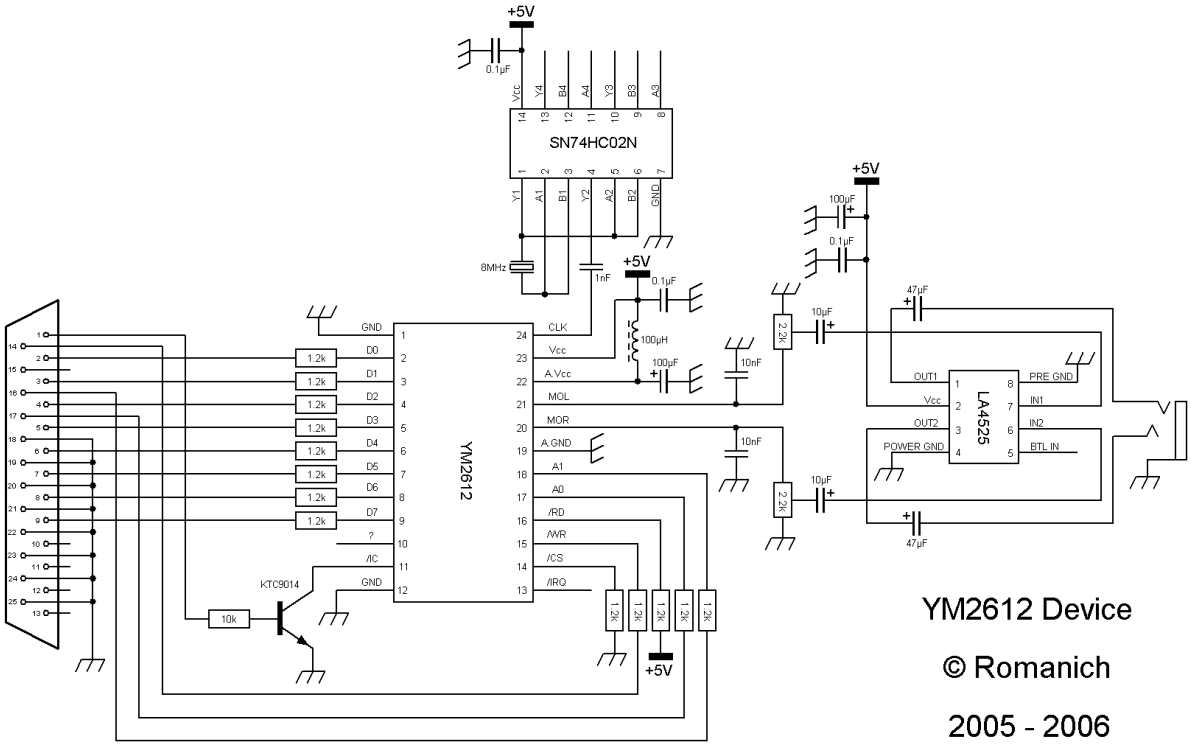
Звуковой чип не имеет встроенного тактового генератора, поэтому просто подсоединить кварц в ножке CLK нельзя! На микросхеме SN74HC02N собран тактовый генератор на 8 МГц. Функционально — это генератор, состоящий из вентиля ИЛИ-НЕ, охваченного кварцем по обратной связи и буферный каскад на следующем вентиле ИЛИ-НЕ.

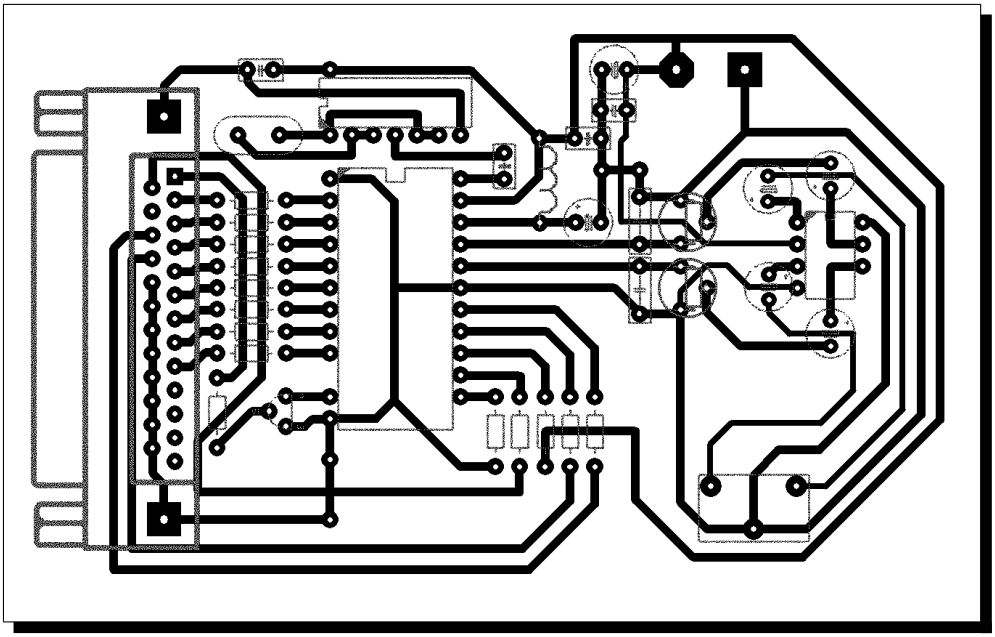
Звуковой чип ещё хорош тем, что он имеет встроенный ЦАП, поэтому с ножек MOL (левый канал) и MOR (правый канал) аналоговый сигнал поступает на вход двухканального (стереофонического) усилителя звука LA4525, который почти не имеет никакого обвеса (кроме разделительных и фильтрующих конденсаторов) и достаточно линеен.

Интерфейс, как уже было сказано выше — порт LPT. Данный порт хорош тем, что обеспечивает удобное управление внешними устройствами (периферией) с помощью компьютера. Причём все его выходные линии (12 штук) используются: 8 бит данных, RES, WR, A0, A1.

Ключ, собранный на транзисторе KTC9014, нужен для корректного сброса звукового чипа (вывод /IC внутрисхемно установлен в "1", т.е. он подтянут).

[4] E-mail автора: romanichapparate@mail.ru


+5V
SN74HC02N
YM2612
LA4525
KTC9014
8MHz
YM2612 Device
© Romanich
2005 - 2006



Питание +5V берётся от USB-порта в компьютере. Но предпочтительнее (для избежания помех по питанию при активной работе блоков компьютера) использовать внешний блок питания на напряжение 5В (ток отдачи не менее 500 мА). Перечень элементов:

Элемент	Кол-во	Примечание
YM2612	1	Аналоги: PCS8593, TA-07
SN74HC02N	1	
LA4525	1	
KTC9014	1	
8MHz	1	
DRB-25MA	1	Разъём LPT порта со штырьками, паяется к плате
Audio	1	Разъём под штекер для наушников, колонок, усилителя
USB	1	Разъём для подачи напряжения питания 5В от USB
1.2 кОм	13	
10 кОм	1	
2.2 кОм	2	Подстроечные резисторы
0.1 мкФ	3	
10 нФ	2	
1 нФ	1	
100 мкФ	2	
47 мкФ	2	
10 мкФ	2	
100 мкГн	1	
DIP24	1	Панелька для YM2612, широкая
DIP14	1	Панелька для SN74HC02N
DIP8	1	Панелька для LA4525

Рисунок печатной платы приведён на предыдущей странице. Как видно из рисунка — плата односторонняя, имеет две перемычки, монтаж довольно неплотный, поэтому трудностей при изготовлении печатной платы не должно возникнуть. Плату легко подготовить в домашних условиях с помощью перманентного маркера или утюжной технологией.

Программирование чипа

Прежде чем рассказать о программировании YM2612, давайте подробно рассмотрим назначение его выводов. Краткое назначение каждой ножки:

GND — цифровая земля (”минус” напряжения питания);

D0 . . . D7 — шина данных (для установки адреса регистра и его значения);

? — вывод неизвестного назначения (оставить неподсоединенным);

/IC — вывод сброса микросхемы (обнуляются все регистры), внутрисхемно уже установлен в ”1”;

CLK — подача сигнала 8 МГц (уровень КМОП или ТТЛ, наличие постоянной составляющей крайне нежелательно, т.е. подавать через конденсатор);

Vcc — подача напряжения питания +5В для цифровой части звукового чипа;

A.Vcc — напряжение питания +5В для аналоговой части звукового чипа;

MOL — аналоговый выход левого канала (открытый эмиттер);

MOR — аналоговый выход правого канала (открытый эмиттер);

A.GND — аналоговая земля (?минус? питания);

A0,A1 — шина адреса (A0 — адрес регистра/данные, A1 — каналы 1 . . . 3 / 4 . . . 6);

/RD — строб чтения (в нашем случае установлен постоянно в ”1” — т.е. чтение не используется);

/WR — строб записи (запись происходит когда установлен в ”0”);

/CS — выбор кристалла (в нашем случае установлен постоянно в ”0” — т.е. микросхема всегда выбрана);

/IRQ — для реализации системы прерываний (см. Описание YM2612), у нас не используется.

Фактически, в нашем случае для управления чипом нужны выводы D0 ... D7, /IC, /WR, A0, A1. Я лишь только порекомендую обратиться к описанию производителя (datasheet) на YM2203 с целью, чтобы подробнее ознакомиться с ним, так как он аналогичен нашему чипу. Описания на YM2612 не доступно в свободном виде, так как это — собственность $SEGA^{TM}$ и в данный момент по интернету не удается найти что-нибудь про него, связанное с железом, кроме самой SEGA. Есть только описание регистров (что собственно совпадает с YM2203). Были найдены принципиальные схемы SEGA MegaDrive (aka Genesis). Именно из них я взял схему включения YM2612.

А теперь вкратце о самой микросхеме YM2612 с программистской точки зрения. Звуковой чип делится на два банка: условно назовем Bank0 и Bank1. Bank0 отвечает за каналы 1, 2, 3, а Bank1 — за каналы 4, 5, 6. В данном случае — каналов 6, то есть одновременно может звучать 6 инструментов. Каждый канал имеет 4 оператора, которые могут соединяться одним из восьми способов (определены в datasheet), создавая при этом различные схемы (алгоритмы) FM-синтеза (для реализации разных классов музыкальных инструментов).

Если нужно работать с Bank0, то перед использованием регистров этой части (каналы 1-3), нужно, чтобы было A1=0. Если необходимо задействовать Bank1 (каналы 4-6), то A1=1. То есть нужная часть звукового чипа определяется посылкой в A1 нуля или единицы — микросхема как бы внутренне состоит из двух частей.

Перед тем как общаться с регистрами YM2612, необходимо её сбросить. Делается это так:

- A1=0 — выбираем Bank0
- IC=0 — /IC=1
- IC=1 — /IC=0 — строб сброса для Bank0
- IC=0 — /IC=1
- A1=1 — выбираем Bank1
- IC=0 — /IC=1
- IC=1 — /IC=0 — строб сброса для Bank1
- IC=0 — /IC=1

После этого все регистры YM2612 в обеих её частях будут обнулены. Обратите внимание, что мы формируем строб RESET (IC) инверсно!!! Это неизбежно, так как к выводу /IC подключен транзисторный ключ, инвертирующий сигнал, поэтому на базу транзистора посылка должна быть такой (IC): 0, 1, 0, а не 1, 0, 1. С коллектора сигнал же будет инвертирован и на вывод /IC пойдёт: 1, 0, 1, т.е. как надо. Сброс делать необходимо, так как при включении микросхемы её регистры запрограммированы случайными значениями — возможна грязь и посторонние звуки.

А теперь об общении с регистрами чипа. Проводится в два этапа — выбор адреса регистра и запись в выбранный адрес значения. Символично это выглядит так:

$$reg[A] = D$$

То есть reg[A] — это регистр с конкретным адресом, D — это его значение. A0 отвечает за смысл данных — если A0=0, то будет записываться АДРЕС регистра, если A0=1, то будут записываться ДАННЫЕ в уже выбранный регистр. /WR — это строб записи, передёргивается в "0", когда адрес/данные надо уже писать и обязательно устанавливается в "1", когда идут манипуляции с A0. Строб чтения /RD должен быть запрещён, т.е. установлен в "1", а микросхема выбрана — /CS=0. D — это байт шины данных (т.е. D0 ... D7). В общем, алгоритм таков:

- /WR=1 — отключаем запись
- A0=0 — будем писать адрес регистра
- D=Address — пишем адрес регистра (т.е. выбираем нужный регистр)
- /WR=0 — включаем запись
- Delay — ждём пока микросхема отработает (для LPT порта не надо, т.к. он медленный)
- /WR=1 — отключаем снова запись (готовимся к формированию данных)
- A0=1 — будем в выбранный регистр писать данные
- D=Data — пишем данные в выбранный регистр
- /WR=0 — активизируем запись

- Delay — ждём...(LPT порт сам по себе медленный, ждать не надо!)
- /WR=1 — запрещаем запись

Упомяну ещё раз один важный момент. Если обращение идет к регистрам чипа каналов 1, 2 или 3 (Part1), то перед процедурой записи обязательно следует указывать, что работа идёт с Bank0 (A1=0). Если нужны каналы 4-6, то A1=1 (Bank1).

Примечание: Вместо сброса YM2612 можно просто записать нули во все её регистры, но, на мой взгляд, с точки зрения программистской эстетики — это некрасиво!

После того как разобрались с записью в регистры чипа, можно смело его программировать. Самоё первое, что приходит на ум — воспроизвести музыку из игр СЕГА, о чём рассказывается далее...

GYM файлы

Практически любой нормальный эмулятор SEGA MD/Genesis на IBM PC имеет возможность создавать файлы музыки в двух форматах: WAV — это для PC и GYM — это просто дамп регистров звукового чипа YM2612 со служебными данными. Сразу хочу предостеречь, что эмуляторы под Windows создают очень кривые GYMы! Что самое интересное — все плагины (под WinAMP например) или специальные плееры воспроизводят их нормально! А на реальном звуковом чипе при воспроизведении таких GYMов будет смешная какофония! Неудивительно, так как та же ошибка сделанная при записи, дает при чтении верный результат. Поэтому я призываю использовать эмулятор SEGA под MS-DOS (Genecyst), он очень качественно подходит к решению вопросов по созданию GYM-файлов.

А теперь о самом формате GYM. Он предельно прост, компактен и очень хорошо сжимается архиваторами! Речь будет идти сейчас о GYM-формате, не содержащем ничего кроме служебных данных и дампа регистров.

GYM формат состоит из четырёх инструкций: 0, 1, 2, 3.

- Байт=0 — ждать 1/60 секунды.
- Байт=1 — запись в регистры Bank0. Далее адрес регистра и далее его значение.
- Байт=2 — запись в регистры Bank1. Далее адрес регистра и далее его значение.
- Байт=3 — запись в PSG одного следующего байта данных.

PSG - это Program Sound Generator, его обсуждение выходит за рамки этой статьи, поэтому он не обсуждается. В YM2612 его нет (в последних версиях приставок SEGA MD/Genesis он интегрирован в Видео-процессор).

Пример (вымышленный!) *0x00, 0x00, 0x00, 0x01, 0x23, 0x55, 0x02, 0x34, 0xFF, 0x03, 0xFA, 0x00* декодируется так:

- ждать 1/60 сек.,
- ждать 1/60 сек.,
- ждать 1/60 сек.,
- Bank0 reg[0x23]=0x55,
- Bank1 reg[0x34]=0xFF,
- PSG=0xFA,
- ждать 1/60 сек.

Вот собственно небольшая, но исчерпывающая информация про GYM-файлы! Следует отметить, что в 99% случаев в GYM-файлы к сожалению не входит дорожка оцифровки (то что напрямую выводится на DAC), то есть ударных в GYMe не будет. Хотя встречаются качественные музыкальные композиции, которые не используют DAC и при этом звучат мощные басы!

Работа с DAC

В YM2612 есть 8-битный встроенный DAC, который программно доступен. Причём он параллельной загрузки, что делает возможным воспроизводить оцифровки звука, посылая их в LPT порт со скоростью не ниже 44100 Гц (хотя это всё зависит от быстродействия IBM PC и оптимизации программы воспроизведения, и конечно же ограничивается предельными характеристиками микросхемы).

Есть пара регистров: 0x2A и 0x2B. Для того, чтоб работать с DAC напрямую, нужно вначале в регистре разрешения DAC установить старший бит в "1", то есть:

$$reg[0x2B] = 0x80$$

А далее в регистр данных DAC класть оцифровку:

for i:=1 to N do reg[0x2A]=Digital[i]

Вот и всё! И постараться делать цикл как можно быстрее! Все задержки реализуются программно (подбирается на слух, чтоб WAV звучал и не быстро и не медленно).

Примечание. Пока используется DAC, то будет недоступен 6-й канал (Bank1). При этом доступны только каналы 1 ... 5.

VGM файлы

К огромному счастью, есть ещё один формат файла, похожий на GYM, который содержит музыкальные композиции из игр под SEGA MD/Genesis! Это VGM-формат, который может содержать дамп регистров звукового чипа YM2612, а также дорожку для DAC! В отличие от GYM-формата, VGM-формат более качественно и экономично подходит к хранению музыкальных композиций. Например, DAC-дорожки (данные PCM) хранятся в единственном экземпляре, а в музыкальном потоке даны лишь ссылки на эти дорожки. А регистры звукового чипа могут при необходимости заполняться со скоростями выше, чем 50/60 Гц, что, несомненно, повышает качество и неотличимость звучания VGM-файла от звучания на реальной игровой приставке! Следует отметить, что VGM-файлов нет как таковых. Есть файлы с расширением VGZ, которые, по сути, являются сжатыми VGM-файлами! Для того чтобы из VGZ получить VGM - можно поступить так: VGZ-файл переименовывается в файл с расширением *.RAR, далее разжимается (например программой WinRAR 3.20) и переименовывается в *.VGM.

А теперь рассмотрим кратко спецификацию VGM (v1.50), то есть здесь объяснены только самые необходимые вещи, которые понадобятся для написания простейшего VGM-декодера.

В самом начале VGM-файл содержит 64-байтный заголовок, с различной информацией о нём. Нас интересуют следующие байты:

1. Двойное слово (4 байта) по адресу 0x1C - это смещение точки цикла. Оно равно нулю, если зацикливание не предусмотрено или содержит значение, к которому надо прибавить 0x1C (чтобы получить абсолютное (от начала файла) смещение точки цикла). После окончания звукового потока (команда 0x66) следует перейти к смещению точки цикла (то есть - прыгнуть на это смещение, по идее близко от начала музыкального потока).
2. Двойное слово по адресу 0x34 - начало музыкального потока, к которому надо прибавить 0x34 чтоб вычислить абсолютный адрес потока в VGM-файле.

Музыкальный поток содержит команды и данные:

- Команды 0x30 ... 0x50 пропускаются, при этом указатель музыкального потока каждый раз увеличивается на 2.
- Команды 0x51, 0x54 ... 0x5F, 0xA0 ... 0xBF пропускаются, при этом указатель потока увеличивается на 3.
- Команды 0xC0 ... 0xDF пропускаются, указатель потока увеличивается на 4.
- Команды 0xE1 ... 0xFF пропускаются, указатель увеличивается на 5.

Далее рассмотрим команды, касающиеся только YM2612, DAC, временные задержки и позиционирование.

- 0x52, [A], D - запись в регистр с адресом A данных D (Bank0). Указатель увеличивается на 3.
- 0x53, [A], D - запись в регистр с адресом A данных D (Bank1). Указатель увеличивается на 3.
- 0x61, nn, nn - задержка nnnn сэмплов (nnnn=0 ... 65535). Один сэмпл равен 1/44100 секунд. Указатель музыкального потока увеличивается на 3.
- 0x62 - задержка на 1/60 секунды (735 сэмплов). Указатель потока растёт на 1.
- 0x63 - задержка на 1/50 секунды (882 сэмпла). Указатель потока растёт на 1.
- 0x66 - конец музыкального потока: это значит либо мы можем прекратить воспроизведение или перейти в точку зацикливания (см. выше). Или начать играть музыкальную композицию сначала? Указатель

музыкального потока устанавливается в первоначальное положение (начало / зацикливание).

- 0x67, 0x66, tt, ss, ss, ss, ss - конкретно: 0x67 - маркер блока данных для DAC; 0x66 - конец потока (здесь эта под-команда игнорируется!); tt - тип данных (должно =0 - это значит YM2612 PCM data); ssssssss - объём данных для DAC в байтах. Здесь логично запомнить смещение данных для DAC, тоесть начальный указатель на PCM data в VGM-файле. Указатель музыкального потока увеличивается на ssssssss+7.

- 0x7* - задержка на *+1 сэмплов. Напомню, что 1 сэмпл = 1/44100 секунды. Указатель потока увеличивается на 1.

- 0x8* - конкретно: A1=0 - выбираем Bank0; reg[0x2A]=Digital[Offset]; Задержка на * сэмплов; Увеличиваем Offset на 1 (указатель на PCM data); Увеличиваем указатель потока на 1. Попросту говоря, данная команда пихает в DAC один байт PCM-данных (указатель на PCM вычисляется командами 0x67 и 0xE0). Далее задержка на * сэмплов и увеличение указателей музыкального потока и PCM data на 1.

- 0xE0, dd, dd, dd, dd - вычисление увеличения указателя на PCM data. Итоговое значение указателя PCM data в данный момент: Offset=ddddddddd+Адрес начала данных для DAC в VGM-файле (адрес байта, следующего сразу после команды 0x67, 0x66, tt, ss, ss, ss, ss). Указатель музыкального потока увеличивается на 5.

Оцените высокое качество воспроизведения VGM-файлов по сравнению с GYM-файлами, а также наличие полноценного DAC. Конечно, для 100%-ной схожести не хватает PSG, но этот недостаток более незаметный, чем отсутствие DAC-дорожки в GYM-файле. Следует отметить, что декодер VGM-файла получается намного сложнее, чем декодер GYM-файла, но не напрасно!

Patches

С большим трудом мои поиски в Интернете увенчались успехом! На сайте http://www.valsound.com/ мне удалось откопать несколько патчей для OPN2 чипов. По сути - это похоже на General MIDI Instruments для Adlib, но только для OPN2. Автором патчей оказался Takeshi Abo, который выложил их в разделе FM-Sound Library.

Разобравшись с форматом данных патчей, я приступил к их обработке (адаптировал их для YM2612). И вот, после нескольких дней, благодаря моему упорству, написаны программы OPN2.exe и OPN2P.exe [5].

Без этих программ пришлось бы довольствоваться проигрыванием мелодий с приставки SEGA и/или слушать 8-битные оцифровки. А так появляется возможность подобрать 258 разнообразных музыкальных инструментов и звуковых эффектов из следующих групп:

- Bass
- Bell
- Brass
- Guitar and Distortion
- Lead-Synth and Organ
- Percussion
- Piano
- Sound Effect
- Special
- Strings and PAD
- Wind
- World

То есть, OPN2 Patches - программа, которая позволяет экспериментировать со встроенными в неё патчами - менять их параметры (Блок, Частоту) случайным образом (OPN2.exe) или вручную (OPN2P.exe).

Основная задача программы OPN2 Patches - это помочь найти нужные звуковые эффекты, подобрать их параметры (Блок, Частота) и далее - использовать патчи в своих разработках! В поставке программы есть файл патчей

[5] Для получения этих программ, обращайтесь к автору по электронной почте — romanichapparate@mail.ru

(бинарник) - OPN2. Ниже приведён формат патчей, лежащих в OPN2.

Основные положения:

- всего 258 патчей, нумеруемых с 0 до 257
- каждый патч описывается 38 байтами
- лишних данных в файле нет, поэтому его объём 9804 байта

Формат:

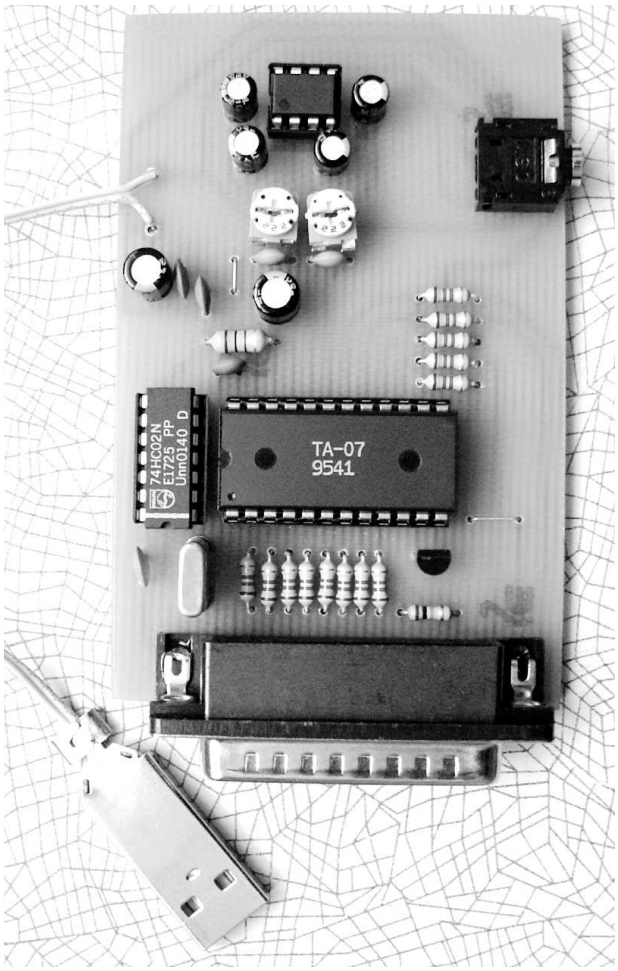
AL FB
AR1 DR1 SR1 RR1 SL1 OL1 KS1 ML1 DT1
AR2 DR2 SR2 RR2 SL2 OL2 KS2 ML2 DT2
AR3 DR3 SR3 RR3 SL3 OL3 KS3 ML3 DT3
AR4 DR4 SR4 RR4 SL4 OL4 KS4 ML4 DT4

AL - Algorithm
FB - FeedBack
AR* - Attack Rate
DR* - Decay Rate
SR* - Sustain Rate
RR* - Release Rate
SL* - Sustain Level
OL* - Operator Level
KS* - Key Scale
ML* - Multiple
DT* - Detune

Где * - номер оператора = 1...4

Описание данных программ завершает эту статью. Считаю, что в этой статье исчерпывающе рассказано о возможностях и сферах применения звукового чипа YM2612. Особо полезной эта статья будет для тех людей, которые самостоятельно проектируют свои вычислительные системы - микрокомпьютеры, игровые приставки и прочие маломощные мультимедиа- девайсы! Подключение цифровых устройств к LPT-порту весьма наглядно демонстрирует их возможности и алгоритмы управления!

Творческих успехов Вам в программировании внешней звуковой FM-карты! Не дадим исчезнуть навсегда железному FM синтезу!



Ссылки

- Эмулятор Genecyst:
  http://www.zophar.net/genecyst/
- Плеер Meridian Advance:
  http://meridian.overclocked.org/
- Плагин WinAmp для GYM:
  http://www.kysoft.net/development/winamp/plugins/in_gym/
- Datasheet YM2203:
  http://www.funet.fi/pub/msx/mirrors/msx2.com/vortexion/ym2203.pdf

Достаточно одной инструкции

Автор: Mac Buster[6]

Название каких микропроцессорных архитектур тебе известны, уважаемый читатель? **CISC** — *Complex Instruction Set Computer? Хорошо. Видно, что ты знаком с историей микропроцессорной техники.* **RISC** — *Reduced Instruction Set Computer? Да, этот термин в последнее время очень популярен.* **VLIW** — *Very Long Instruction Word? Замечательно! Ты, вероятно, прочёл или прослушал курс по разработке высокопроизводительных микропроцессорных систем.* **EPIC** — *Explicitly Parallel Instruction Computer? Прекрасно! Похоже ты всерьез задумывался о возможных путях развития этой области. Что ещё можно упомянуть? Пожалуй стоит вспомнить* **MISC** — *Minimal Instruction Set Computer или микропроцессор с минимальной системой команд. Например, мне известно о работах по реализации микропроцессора (Форт-системы), в систему команд которого входит всего-навсего 7 (семь) инструкций, и, как ни странно, этого вполне достаточно для выполнения практически любой задачи, которая может быть поставлена перед системным программистом. Считаешь сомнительным, что с помощью всего семи инструкций можно вообще решить какую-нибудь задачу? Не уверен? А зря! Такие процессоры существуют уже давно, и применяются в авиационных и космических системах. А что ты подумаешь, если я скажу, что существует архитектура, в которой всего...одна команда! Нет, я не шучу, такая архитектура в самом деле есть. Называется она на удивление предсказуемо:* **OISC** — *One Instruction Set Computer, или еще короче — просто OIC.*

Кажется невероятным, однако, для решения любой задачи действительно требуется одна и только одна команда! Ещё раз повторю что это не шутка. Более того, у этого процессора нет ни одного регистра или флага состояния. Впервые об этой архитектуре я узнал из документа написанного и распространённого сотрудником корпорации Hewlet Packard по имени Ross Cunniff из лаборатории по разработке программного обеспечения для работы с графическими данными. Причём он не ограничился одной лишь теорией, а написал компилятор и интерпретатор для выполнения программ, написанных на ассемблере этого удивительного, и, вероятно, ещё не существующего в реальности процессора.

Какой волшебной командой можно успешно решить любую задачу? Вот такой - **Subtract and jump if negative** (вычесть и перейти, если результат операции отрицательный). У этой инструкции три указываемых через пробел операнда:

X Y Z

Первый из них — X — адрес ячейки в которой находится уменьшаемое, туда же сохраняется результат вычитания. Второй — Y — адрес ячейки в которой находится вычитаемое. Третий — Z — номер ячейки которой должно быть передано управление в том случае, если в результате вычитания Y из X получится отрицательный результат. Если разность неотрицательна (положительна или равна нулю) управление будет передано инструкции следующей за текущей. В этом прослеживается некоторое отдаленное сходство с архитектурой процессоров управляемых потоком данных. Как ты, возможно, уже догадался, нам нет необходимости записывать код операции - КОП, поскольку она у нас одна-единственная, достаточно записывать только фактические операнды. Давай для лучшего понимания рассмотрим работу OISC-процессора на нескольких примерах.

[6] E-mail автора: mbr@ternary.info

Пример 1.
Пусть при выполнении программы процессор выбрал из памяти такую команду:

1 2 16384

И при этом в ячейке номер 1 содержится уменьшаемое — число 123, а в ячейке с номером 2, вычитаемое — число 321. В результате выполнения вычитания получится разность 198, которая будет помещена в память на место уменьшаемого. При этом, так как результат положительный, управление будет передано следующей по порядку инструкции. В противном случае управление было бы передано инструкции расположенной по адресу 16384.

Пример 2.
При выполнении следующей команды:

1 2 4096

Где в ячейке номер 1 находится число 5 (уменьшаемое), а в ячейке номер 2 — число 3 (вычитаемое). В результате вычитания получится разность -2 (минус два), — отрицательное число, которое будет записано в память по адресу уменьшаемого. Следовательно, после вычитания будет выполнена команда расположенная по адресу 4096.

Надеюсь, после ознакомления с приведенными примерами тебе стало понятнее, как работает этот процессор. А чтобы совсем не осталось сомнений, посмотри на псевдокод реализующий эту команду:

```
int program_counter = 0;
int memory[384];
while (TRUE)
{
      a = memory[program_counter];
      b = memory[program_counter + 1];
      c = memory[program_counter + 2];
      memory[a] = memory[a] - memory[b];
      if (memory[a]>0)
            program_counter += 3;
      else
            program_counter = c;
}
```

Теперь настало время рассказать о том, каким образом с помощью одной команды можно писать программы выполняющие что-то кроме вычитания двух чисел. Секрет, как обычно, прост — компилятор позволяет задавать макроопределения, что при некоторой сообразительности и знании того, что такое самомодифицируемый код позволит вам реализовать любую функцию и затем просто использовать её в любой момент в необходимом месте программы, которая, в свою очередь, уже почти полностью состоит вызовов макросов. При использовании макроопределений часто возникает необходимость знать текущий адрес компилируемой инструкции. В качестве относительного указателя текущего адреса компиляции используется символ "." (точка), за которой следует число, указывающее компилятору сколько команд (слов, ячеек) после текущей следует пропустить. Перед числом принято ставить плюс или минус, для указания направления перехода — вперед — в сторону увеличения адресов, либо назад — в сторону уменьшения.

Впрочем, автор уже позаботился о тебе и в архив с компилятором, интерпретатором и кратким описанием архитектуры включил файл содержащий некоторые из часто требующихся макроопределений.

Поняв всё написанное выше можно попытаться реализовать некоторые операции микропроцессора традиционной архитектуры. Например, реализация команды безусловного перехода совсем проста и выглядит следующим образом:

Jmp C = Z Z .+1

Где Z — предварительно зарезервированная ячейка памяти содержащая нулевое значение.

Рассмотрим ещё несколько примеров реализации инструкций. Обратите внимание, что теперь я не буду указывать третий операнд — адрес перехода, если он указывает на следующую по порядку инструкцию. Сложить два числа можно с помощью такой конструкции:

ADD a,b = a, Z
Z, b
Z, Z

Первая инструкция производит смену знака одного из слагаемых — a — и сохраняет полученное число в ячейке Z. Второй инструкцией

производится вычитание первого слагаемого -a из сторого слагаемого b. Третья инструкция обнуляет содержимое ячейки Z с тем, чтобы при следующем использовании этого набора они отработали правильно.

Операция копирования содержимого одной ячейки в другую можно выполнить таким образом:

```
STO a,b = b, b
          a, Z
          Z, b
          Z, Z
```

Как видишь, эта конструкция практически полностью повторяет конструкцию, которую мы рассмотрели выше.

Рассмотрим набор инструкций реализующих операцию сравнения и перехода по заданному адресу в том случаеЮ если результат сравнения равен нулю:

```
BEQb,c = b, Z, L1
         Z, Z, OUT
    L1:  Z, Z
         Z, b, c
    OUT:...
```

Как видите, компилятор OISC, как и всякий другой приличный язык, предоставляет программисту возможность задавать символьные имена для ячеек, т.е. определять метки. Имя метки может быть любым, но должно отличаться от десятичного числа и обязательно оканчивается символом ":" (двоеточие).

Кроме этого в синтаксисе имеется возможность явно разделить текст на логические блоки по их назначению: для кода (служебным словом **TEXT:**) и для данных (словом **DATA:**).

Для взаимодействия с внешним миром интерпретатор имеет три ячейки памяти с жёстко заданным назначением. Одна из них предназначена для чтения данных из файла или с консоли машины на которой выполняется интерпретатор. Она имеет десятичный адрес 32765. Вторая используется для выдачи значений в файл или на консоль. Ее адрес - 32766. Обратите внимание, что будет выдано число с противоположным знаком! Попытка записи или чтения значения из третьей ячейки или при использовании её в качестве адреса перехода после выполнения команды приводит к остановке интерпретатора. Этой ячейке соответствует адрес 32767. Для простоты программирования им присвоены символьные имена: **READ:**, **WRITE:** и **STOP:** соответственно.

На этом я завершаю описание архитектуры и хочу сказать на прощание несколько общих слов.

1. Собственно говоря, этот процессор демонстрирует практическое применение логической функции ИЛИ-НЕ, с помощью которой, как известно из математической логики, можно реализовать любую другую логическую функцию.

2. Справедливости ради следует отметить, что OISC не единственный представитель класса микропроцессорных архитектур основанных на выполнении одной-единственной команды. Кроме нее есть существуют следующие: uRISC (ultimate RISC), а также целый ворох так называемых move-машин, по сути представляющих собой реализацию машины Поста. Однако, на мой взгляд, OISC значительно проще в понимании и в программировании, т.к. непосредственно выполняет одно из основных арифметических действий и сохраняет результат в памяти, что позволяет реализовать остальные арифметические операции не прибегая к помощи промежуточных таблиц хранящихся в ОЗУ.

3. Только настоящий мастер своего дела не боится трудностей и может произвести на свет шедевр, используя инструмент с самыми скромными характеристиками ;)

Попробуйте свои силы! Желаю вам успеха!

P.S. Пока верстался номер стало известно о выпуске фирмой Dallas/Maxim Semiconductors целой серии микроконтроллеров с архитектурой реализующей один из возможных вариантов OISC. В данном случае микроконтроллер выполняет инструкцию MOVE.

PENTAGON 1024SL V2.2

Основные технические характеристики платы ver 2.2:

Конструкция платы соответствует AT - стандарту

Архитектура открытая (3 слота ZX-BUS)

Тип процессора KP1858BM3 / Z0840008PSC

Тактовая частота 7 МГц (TURBO) / 3.5 МГц (NORMAL)

Объем ОЗУ 1024 Кб

Объем ПЗУ 64 Кб

Интерфейс принтера ZX LPRINT III (реализована только аппаратная часть), совместим с CENTRONICS

Типы джойстиков KEMPSTON, INTERFACE II

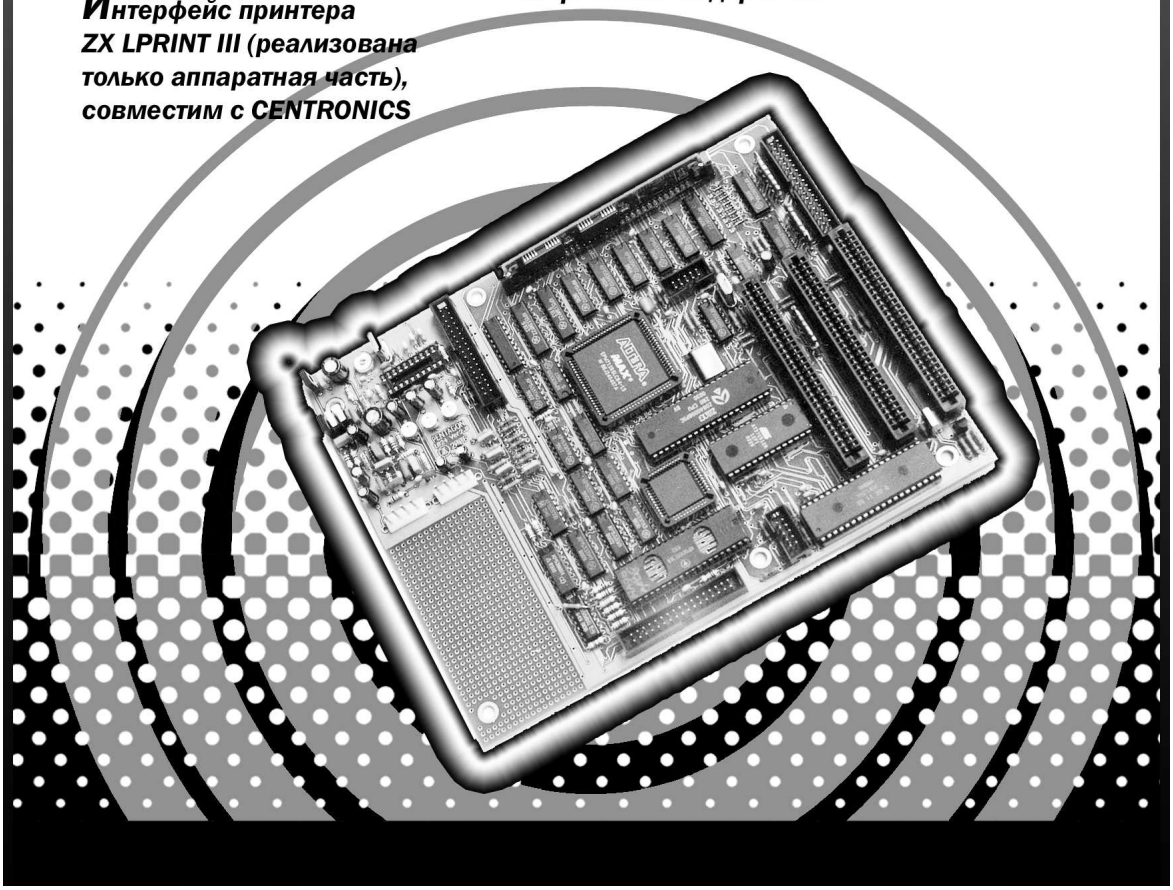
Музыкальный сопроцессор YM2149F/AY8910

Типы видеорежимов стандартный ZX-экран, 16 COLOUR (ZX экран, каждая точка рисуется своим цветом), ZX-экран без бордюра (разрешение 384 x 304)

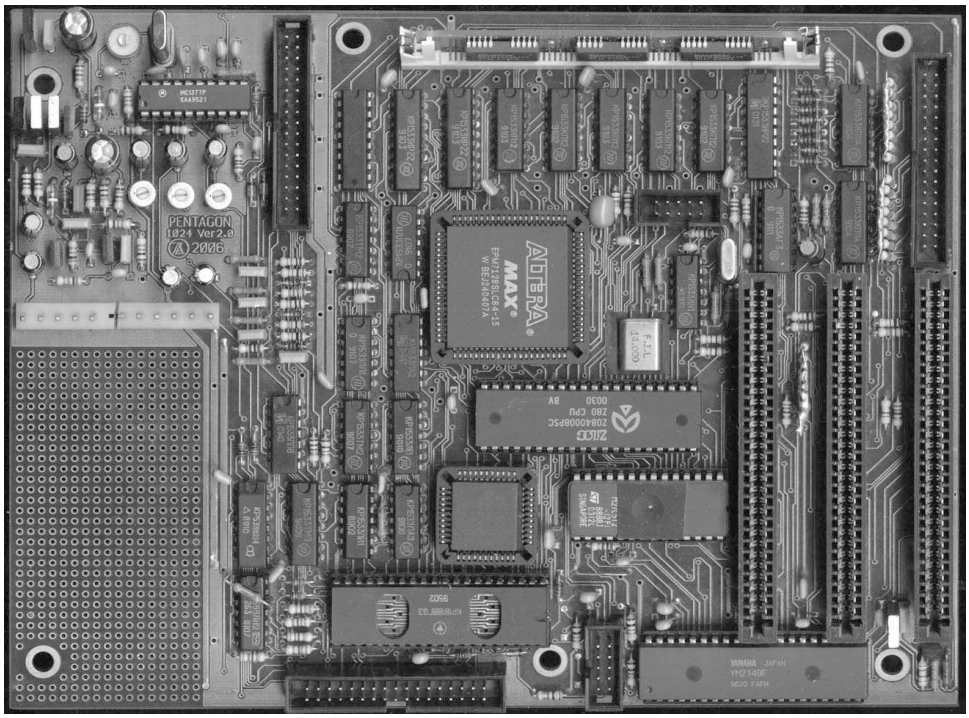
Контроллер дисковода BETA DISK, на базе KP1818ВГ93

Звуковой усилитель 2 x 0.5 Вт

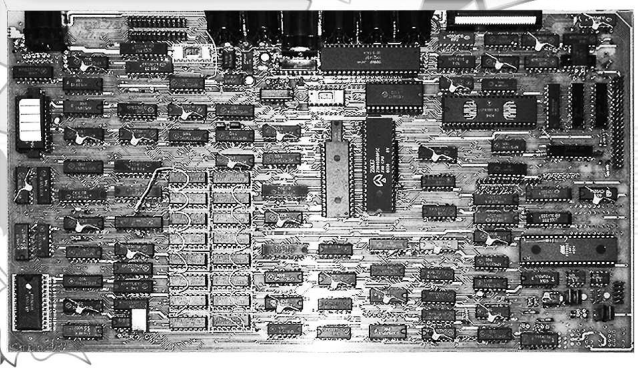
Подключение к монитору / TB RGB SYNC (нагрузка 75 Ом, размах 1.5 В пик-пик), встроенный кодер NTSC



Pentagon-1024SL



Позиция	Цена, руб.	Замечания
Собранная и настроенная печатная плата (ver 2.2), полный комплект технической документации (CD-ROM, содержащий принципиальную и монтажную схемы, краткое описание схемы и инструкцию по сборке и настройке, а также архив с разной полезной информацией, литературой, программами в TRD формате и т.д.), 2 шлейфа для внешней периферии (монитор, клавиатура, джойстик и т.д.) — с одной стороны розетка IDC-34. Набор разъёмов для внешней периферии.	3000	В комплект не входят SIMM модуль и музыкальный процессор AY-3-8910 / YM2149F.
Пустая печатная плата (ver 2.2), полный комплект технической документации (CD-ROM, содержащий принципиальную и монтажную схемы, краткое описание схемы и инструкцию по сборке и настройке, а также архив с разной полезной информацией, литературой, программами в TRD формате и т.д.).	800	Платы делались по 3-му классу точности на заводе в Зеленограде, имеется защитное масочное покрытие.
Пустая печатная плата (ver 1.4), полный комплект технической документации (CD-ROM, содержащий принципиальную и монтажную схемы, краткое описание схемы и инструкцию по сборке и настройке, а также архив с разной полезной информацией, литературой, программами в TRD формате и т.д.).	600	Платы делались по 3-му классу точности на заводе в Зеленограде. Масочное покрытие отсутствует.



АТМ Турбо

от NedoPC

Позиция	Цена	Примечания
Плата голая Комплектность поставки: - плата; - ХЛ8 (прошитая); - описание (книжки); - софт (имиджи на сдром).	700	Платы при изготовлении электрически не проверяются
Плата собранная Комплектность поставки: - отлаженная плата с 1556ХЛ8, ПЗУ, Z80, 1818ВГ93, панелька под AY; - описание (книжки); - софт (имиджи на сдром)	2950	Платы будут собраны и отлажены, т.е гарантировано рабочие. Музыкальный сопроцессор не поставляется
Конструктор для самостоятельной сборки: Комплектность поставки: - плата; - набор радиодеталей, необходимых для сборки; - необходимые ПЗУ (прошитые), ХЛ8; - описание (книжки); - софт (имиджи на сдром).	2300	Процессор, ОЗУ и ПЗУ проверяются перед отправкой. Поставщик не несет ответственности за сгоревшие в результате неправильной сборки или подключения детали
Комплект шлейфов: - шлейф FDD - шлейф IDE - шлейф COM - переходник AT-питание->5-DIN - шлейф для звука	200	Комплект облегчает установку в AT корпус
Музыкальный сопроцессор (YM2149)	160	
Прошивка ПЗУ основная (27С512 или аналог)	65	На текущий момент версия 1.07.13
Прошивка тест ОЗУ (27С512 или аналог)	65	На текущий момент версия 1.02
Прошивка знакогенератора (573РФ2 или аналог)	35	Символьная таблица знакогенератора
Прошивка контроллера клавиатуры (573РФ2 или аналог)	35	XT или AT клавиатура

К общей стоимости заказа добавляется цена доставки (по России):

130 руб. для голой платы

200 руб. для платы в сборе

Порядок обработки заказов на платы:

1. Присылаем e-mail Чунину Роману Валерьевичу (chunin@mail.ru) или звоним по телефону +7(095)6547433, для выяснения есть ли свободные платы и согласования цены, комплектации;
2. Если не получен почтовый перевод в течении двух недель, то заявка снимается;
3. Дополнительно к стоимости добавляется стоимость услуг почты: 130 руб. для голой платы, 200 руб. для собранной;
4. Осуществляем почтовый перевод на адрес: 109451, Москва, ул.Братиславская, д.13, кор.1, кв.228, Чунину Роману Валерьевичу. Пожалуйста, указывайте обратный адрес (если через e-mail, то с указанием номера и даты почтового перевода);
5. После обработки и подготовки заказ отсылается по указанному вами адресу. Голые платы отсылаются в течении одной недели после оплаты, собранные в порядке очереди на сборку (на сборку одной платы уходит примерно неделя, собирать будут два человека параллельно);
6. Весь процесс оформления заказов будет отображаться на сайте http://nedopc.com/products.php

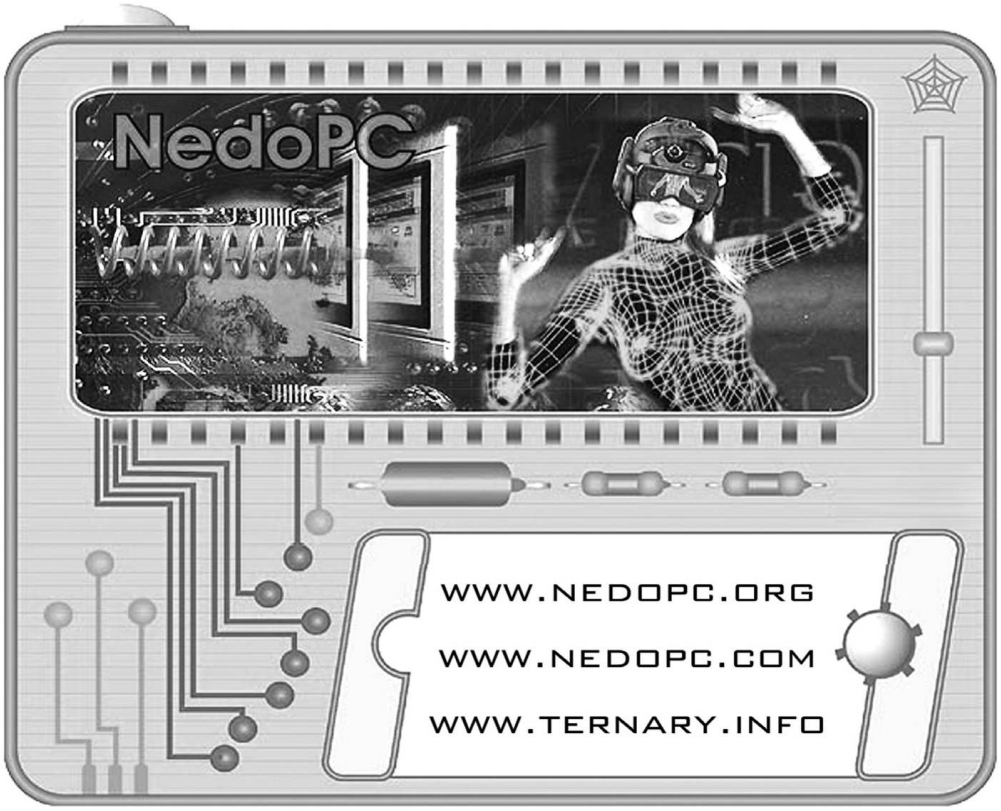
Новости троичной эмуляции

Уважаемый читатель, спешим поделиться с тобой хорошими новостями из области исследования уравновешенной троичной системы счисления. Не успел еще разойтись предыдущий выпуск нашего издания, как были завершены работы по созданию предварительных версий веб-симулятора ЭВМ "Сетунь" - Setun59ws и платформы эмуляции троичного процессора и ЭВМ - "TVM", а так же еще одного оригинального варианта часов, отображающих время в уравновешенной троичной системе счисления! Кратко расскажем о каждом их них:

Александр Никитич из Челябинской области написал на PHP веб-симулятор ЭВМ "Сетунь" основываясь на доступных в интернете документах, а так же на предоставленной мной книге "Малая цифровая вычислительная машина Сетунь". Этот симулятор выполняет практически все команды ЭВМ, за исключением команд обращения к магнитному барабану. Симулятор можно опробовать в действии на нашем сайте: http://ternary.info/setun

Также Александр Никитич сделал флэш-версию троичных часов. Они показывают время не только в виде цифр, но и по разрядам. Благодаря чему их можно использовать как своеобразный тренажер для перевода чисел из уравновешенного троичного представления в десятичное. Посмотреть троичные часы, и почитать интересную статью о представлении информации о текущем времени в уравновешенной троичной форме можно на сайте Новой Эры, по следующему адресу: http://neutec.narod.ru/

Роман Чунин написал на языке Java расширяемый эмулятор троичной машины под рабочим названием "TVM", который может быть использован для исследований эффективности троичных алгоритмов обработки данных. Внешне он представляет собой среду напоминающую отладчик. Программист может видеть и, по своему усмотрению изменять содержимое 27 регистров общего назначения; содержимое оперативной памяти (на данный момент ее объем составляет 19 тысяч троичных слов, по 9 разрядов каждый) и, в будущем, содержимое программного стека. Для удобства работы имеется встроенный калькулятор, с помощью которого можно переводить числа между системами счисления. Интересной особенностью этого эмулятора является то, что каждый желающий сможет придумать и реализовать свою собственную систему команд, что в будущем может привести к интересному соревнованию между разными программистами, которое даст нам наиболее оптимальный набор команд для троичного компьютера. Этот эмулятор можно найти на нашем сайте: http://www.ternary.info



ВСЕГДА В ON-LINE!