

Тегированная память ([tagged architecture](#)) даёт экзотическую возможность отделить данные от метаданных. Цена за это не столь уж и велика (на первый взгляд), а потенциальные возможности впечатляют.

## История вопроса

В тегированной архитектуре каждое слово памяти сопровождается тегом - одним или несколькими разрядами, описывающими данные в этом слове. Сама по себе идея эта очень естественна и возникла на самой заре компьютерной индустрии.

### Rice computer

Хронологически, по-видимому первым полноценным компьютером с такой архитектурой был [R1](#), который разрабатывался с 1958 по 1961 годы, первые признаки жизни стал подавать в 1959. В основном Rice computer запомнился тем, что его память была собрана на [CRT трубках](#).

Аппаратное слово (коих было 32К, сгруппированных по 64) имело размер 64 разряда, из которых:

- 1 разряд использовался для отладки работы самих трубок, которые были весьма ненадёжным хранилищем. Фактически, каждая 64-я трубка была CRT монитором, на котором можно было буквально *подглядеть* содержимое любой из соседних трубок
- 7 разрядов на [код Хэмминга](#)
- 54 разряда собственно данных
- 2 разряда на тег. Теги распространялись на слова кода также как и на данные.

Теги на коде использовались для отладки, например, инструкция с тегом "01" вызывала печать содержимого регистров.

Для данных всё сложнее. Теги использовались для маркировки массивов, например, у двумерных массивов были теги конца строки и конца массива. Также тег мог означать конец итераций (вектора), когда инструкцию циклически напускали на вектор.

### Burroughs Large Systems

Burroughs 5000 появился в 1961г. Каждое 48-разрядное слово сопровождал тег из одного разряда, который определял, код это или данные.



Фиг.2 Некоторое количество машин было поставлено в СССР, породив в дальнейшем слухи, что линейка Эльбрусов - это переделанные Burroughs, что, конечно же, не так.

В дальнейшем (1966, В6500) применение тегов было признано весьма успешным, и тег был расширен до 3 разрядов. При этом 48-й разряд был неизменяемым и по прежнему отделял код от данных. Теги означали:

- 0 - любые данные кроме double precision
- 2 - double precision
- 4 - индекс для циклов
- 6 - неинициализированные данные
- 1 - адреса на стеке
- 3 - общий тег для кода
- 5 - дескрипторы, описывающие данные **не** на стеке
- 7 - дескриптор процедуры

Использовалось это для контроля во время исполнения. Так, невозможно случайно во время исполнения изменить код. Или использовать неинициализированные данные. Невозможно случай но или намеренно подменить адрес возврата из функции и т.д.

Данная серия была очень успешной и коммерчески с точки зрения развития технологий. Единственным серьезным недостатком можно считать сильную ориентацию на [Algol](#). С устареванием Алгола ушел и Burroughs.

### **Lisp - машины.**

Активно развивались в 1973...1987 гг после и во время бума систем искусственного интеллекта на Lisp.

Некоторые из них имели аппаратные теги, например [Symbolics 3600](#) (36-разрядное слово, содержащее 4-8 разрядный тег и 28-32 разряда данных). Здесь теги требуются для описания структуры данных. Имея слово, по его тегу можно понять, что перед нами - просто слово данных или указатель на массив, а может голова списка. В свою очередь, каждый элемент списка может быть или просто словом или указателем на массив или ...

А некоторые использовали технику [тегированных указателей](#), когда тег записывается либо в неиспользуемые биты указателя, либо в память, предшествующую той, на которую смотрит указатель.

Эта техника вполне себе процветает и в наши дни, например, в виде [NSNumber](#). Из близкого автору, подобный lisp-подход (вместе с lisp-ядром) был внедрен сначала в Kuhl, а затем и в [Openlink Virtuoso](#) RDBMS.

### Эльбрус советский.

64-разрядное слово сопровождалось 8 разрядами тега и 8 - кода Хемминга. Развитая система тегов допускала динамическую типизацию. Т.е. имелась общая инструкция сложения, процессор по тегам определял типы аргументов, делал нужные приведения и запускал суммирование нужного типа.

При этом, если аргументом был дескриптор функции, запускалась функция, а если косвенное слово, делалось косвенное обращение. И так до тех пор, пока не получалось именно значение.

Наряду с безадресностью, это обеспечивало высокую компактность кода.

### Эльбрус нынешний (три стека - Sic!)

Эльбрус, насколько известно автору, единственная развивающаяся архитектура с тегированной памятью.

Физически теги расположены в памяти, отведенной под ECC (Error Correcting Code). Свопирование (подкачка) данных в/из виртуальной памяти идёт вместе с тегами, в отличие от обычной ECC памяти.

Каждое 4-байтовое слово в памяти, регистрах и шинах сопровождается 2-разрядным тегом.

Теги бывают -

- дескриптор массива
- дескриптор объекта
- пусто
- числовые данные

Ни длина, ни тип числовых данных архитектурно неразличимы. Семантическое наполнение числовой переменной отслеживается компилятором и проявляется, когда она становится операндом какой-либо операции.

Регистровый файл содержит 256 регистров по 84 разряда каждый, имеющих три поля. Первые два поля предназначены для хранения 32-разрядных слов с 2-разрядными тегами, двойного слова Ф64, мантиссы вещественного числа, а третье поле отводится для хранения 16-разрядного значения порядка. Т.е.  $2 * (32 + 2) + 16$ .

Механизм тегов используется для аппаратного контроля правильности работы программы, издержки составляют 25-30% (forum.ixbt). Издержки, по-видимому, вызваны тем, что указатель может достигать 256 разрядов.

Итого, от динамической типизации в духе советских Эльбрусов отказались, но применили модифицированную объектную модель, которая кроме издержек в производительности, сильно усложнила компилятор C/C++, сделав невозможными попытки встроиться в инфраструктуру [LLVM](#), например.

## **Итак.**

Мы видим, что тегированная память использовалась со следующими целями:

- отладка
- контроль правильности исполнения
- data driven programming - имеется в виду использование данных в (духе LISP), которые сами определяют свою структуру..

Довольно остроумная идея, которая возникла практически вместе с самими компьютерами и чья история еще не закончена. Посмотрим, что будет дальше.