



блог alexanius'a



четверг, 12 мая 2016 г.

Пишем "Hello, world" на ассемблере

Так сложилось, что я совсем не знаю ассемблера. Даже несмотря на то, что я разрабатываю компиляторы, на уровень близкий к аппаратуре я почти не спускаюсь. Была пара попыток его выучить, но я просто не находил подходящего материала. В итоге решил что если его нет, то нужно написать самому. В этой заметке я планирую показать как написать простой Hello world на ассемблере.

В данной статье я преследую несколько целей:

- Изучить основы работы с ассемблером
- Сравнить ассемблеры процессоров различных архитектур и, как следствие, показать разные аппаратные особенности
- Написать материал по которому новички далее смогут самостоятельно продолжить изучение ассемблера

Содержание:

1. Введение
2. amd64
3. sparc v9
4. Эльбрус
5. Послесловие
6. Источники

1. Введение

Я буду стараться давать минимум теории, т.к. её рассказывают много где, гораздо более подробно и понятно. Поэтому буду описывать только то, что касается данного примера. Итак, задача: написать программу, выводящую на экран сообщение "Hello, world". В качестве эталона возьмём программу на C:

```
#include <unistd.h>

int main()
{
    const char * msg = "Hello, world\n";
    write(0, msg, 13);
    return 0;
}

Сборка и запуск:

$ gcc t.c && ./a.out
Hello, world
```

Здесь специально не использована стандартная библиотека, а применён системный вызов `write`. Подробнее про него можно прочесть по команде `man 2 write`.

2. amd64

В качестве процессора на данной архитектуре применяется Intel Core i5, операционная система - Gentoo GNU/Linux, синтаксис AT&T. По моей любимой привычке сначала напишем программу, а потом будем думать.

```
.section .data
hello_str:
.string "Hello, world\n"
.set hello_str_len, . - hello_str - 1

.section .text
.globl _start

_start:
# Здесь подготавливаем и вызываем write
mov $1, %rax
mov $1, %rdi
mov $hello_str, %rsi
mov $hello_str_len, %rdx
syscall

# Здесь подготавливаем и вызываем exit
mov $60, %rax
mov $0, %rdi
```

Ярлыки

[c \(17\)](#) [компиляторы \(17\)](#)
[gcc \(15\)](#) [c++ \(13\)](#) [разное \(13\)](#)
[Эльбрус \(10\)](#) [latex \(8\)](#)
[linux \(7\)](#) [софт \(7\)](#) [gentoo \(6\)](#) [soft \(5\)](#) [статический анализ \(5\)](#) [цитата \(4\)](#)
[basex \(3\)](#) [clang \(3\)](#) [sparc \(3\)](#) [x86 \(3\)](#)
[xqugu \(3\)](#) [МАИ \(3\)](#) [диплом \(3\)](#)
[обучение \(3\)](#) [перевод \(3\)](#) [llvm \(2\)](#)
[osaml \(2\)](#) [pvs \(2\)](#) [а нафига? \(2\)](#)
[ассемблер \(2\)](#) [GeekTime \(1\)](#) [aiSee \(1\)](#) [arduino \(1\)](#) [beamer \(1\)](#) [dmcrypt \(1\)](#) [du \(1\)](#) [fracplanet \(1\)](#) [git \(1\)](#) [html \(1\)](#) [imagine cup \(1\)](#) [lint \(1\)](#) [to \(1\)](#) [man \(1\)](#)
[oricrafter \(1\)](#) [perl \(1\)](#) [prefetch \(1\)](#) [qt \(1\)](#) [saturn \(1\)](#)
[sedna \(1\)](#) [time \(1\)](#) [valgrind \(1\)](#) [windows \(1\)](#) [xvcm \(1\)](#)

Архив блога

► 2020 (1)
► 2019 (1)
► 2018 (1)
► 2017 (2)
▼ 2016 (6)
► июля (1)
► июня (2)
▼ мая (2)
 Профиль программы и
 его предсказание
 Пишем "Hello, world" на
 ассемблере
► января (1)
► 2015 (3)
► 2014 (11)
► 2013 (16)
► 2012 (33)
► 2011 (10)

```
syscall
```

Сборка и запуск:

```
$ as tt.s -o tt.o && ld tt.o && ./a.out
Hello, world
```

Теперь попытаемся понять что произошло.

Краткое описание синтаксиса:

На каждой строке находятся команды (statement). Команда начинается с нуля и более меток, после которых находится ключевой символ, обозначающий тип команды. Всё что начинается с точки `.` является директивой ассемблера. Всё что начинается с буквы является инструкцией ассемблера и транслируется в машинный код. Комментарии бывают многострочными `/**/` и односрочными `#`.

Директивы .section обозначают начало секций. Секция - это диапазон адресов без пробелов, содержащий в себе данные, предназначенные для одной цели [as]. Объектный файл, сконструированный as имеет как минимум три секции: .text, .data, .bss. Внутри объектного файла по адресу 0 располагается секция .text, за ней идёт секция .data, а за ней секция .bss. Все адреса as вычисляют как (адрес начала секции) + (смещение внутри секции). Итак, что же означают секции:

- .data - в этой секции обычно хранятся константы
- .text - в этой секции обычно хранятся инструкции программы
- .bss - содержит обнулённые байты и применяется для хранения неинициализированной информации

В начале секции .data у нас стоит метка hello_str, которая указывает на начало строки.

Далее идёт директива .string. Это псевдо операция, копирующая байты в объектник.

Директива .set присваивает символу значение выражения. Т.о. мы говорим что символ hello_str_len равен выражению . - hello_str - 1. Символ `.` означает текущий адрес. Вычитая из него адрес метки hello_str получаем длину строки с завершающим нулем. Чтобы он не попал на печать вычитаем 1.

Директива .globl говорит что данный символ должен быть виден /d/. Т.е. теперь символ _start сможет быть слинкован. Это нужно, т.к. вход в программу осуществляется именно через этот символ.

После метки _start начинаются непосредственно ассемблерные инструкции. И теперь опять вернёмся к теории.

Данная программа написана под процессор Intel архитектуры amd64 (она же x86_64). Это 64-х битное расширение архитектуры IA-32. Описание самой архитектуры процессора находится в [intel1]. Подробное описание команд процессора находится в [intel2].

Итак, в данной программе мы оперируем регистрами - внутренней памятью процессора. Архитектура amd64 содержит очень мало регистров - всего 16 64-х разрядных регистров общего назначения: RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8D-R15D.

Операция mov предназначена для копирования первого операнда во второй (заметьте, что это особенность синтаксиса AT&T, и интеловский синтаксис имеет обратный порядок операндов). Мы можем скопировать константу, значение общего или сегментного регистра или значение из памяти. Копировать можно в общий или сегментный регистр или память. Для обозначения констант используется символ \$, а для регистров - %. Чуть позже станет понятно что куда и зачем мы копировали.

Далее идёт операция syscall. Она делает системный вызов. Системный вызов - это функция из ядра ОС. Каждый системный вызов производится по номеру. Он должен находиться в регистре rax. Номера системных вызовов можно посмотреть в таблицах [syscall1][syscall2]. Но можно выяснить самому. Их конкретное местоположение зависит от дистрибутива. В моём случае они, например, находятся в файле /usr/include/asm/unistd_64.h. Вот выдержка из этого файла:

```
...
#define __NR_read 0
#define __NR_write 1
#define __NR_open 2
...
#define __NR_execve 59
#define __NR_exit 60
#define __NR_wait4 61
...
```

Понятно, что помимо номеров нам нужны ещё аргументы этих вызовов. Их можно найти следующим образом:

```
$ cd /usr/src/linux/
$ grep -rA3 'SYSCALL_DEFINE.\?(write,' *
fs/read_write.c:SYSCALL_DEFINE3(write, unsigned int, fd, const char
__user *, buf,
fs/read_write.c-          size_t, count)
fs/read_write.c-{           struct fd f = fdget_pos(fd);
```

Но в целом таблицами, подготовленными хорошими людьми пользоваться удобнее.

Итак, видно, что вызов `write` требует 3 аргумента. Первый - это дескриптор файла вывода. Он кладётся на регистр `rdi`. Мы на `rdi` кладём 1, что является дескриптором `stdout`. На регистр `rsi` кладётся указатель на адрес строки. И на регистр `rdx` кладётся длина строки. Всё, теперь, когда все регистры подготовлены, можно делать `syscall` и нам будет выведено сообщение.

Далее нужно выйти из программы. Для этого используется системный вызов `exit`. Он имеет номер 60 и требует код возврата в качестве первого аргумента. Мы завершаемся с кодом 0, как и положено успешно выполненной программе.

3. Sparc v9

Не устали? Теперь внезапно рассмотрим sparc. Меня эта платформа интересует, т.к. одна из линеек процессоров Эльбрус основана на этой архитектуре. Я тестировался на процессорах TI UltraSparc III+ (Cheetah+) с ОС Gentoo и процессорах Эльбрус R1000 с ОС Эльбрус. Итак, смотрим:

```
.section .data
hello_str:
    .ascii "Hello, world\n"
    .set hello_str_len, . - hello_str

.global _start

.section .text

_start:
    ! Подготавливаем и вызываем write
    mov 1, %o0
    set hello_str, %o1
    mov hello_str_len, %o2
    mov 4, %gl
    ta 0x10

    ! Подготавливаем и вызываем exit
    mov 0, %o0
    mov 1, %gl
    ta 0x10
```

Сборка и запуск:

```
$ as -Av9 -64 tl.s -o tl.o && ld -Av9 -m elf64_sparc tl.o && ./a.out
Hello, world
```

Вроде как отличий немного. Синтаксис `as` был описан в блоке `amd64`, разве что здесь односторонние комментарии задаются символом `!`, поэтому его опускаем и переходим сразу к отличиям. Сразу скажу, что речь идёт о Sparc v9 если не оговорено другое. v9 является 64-х битным расширением архитектуры sparc v8. Начнём с регистров. Их здесь больше чем в `amd64` - целых 32 общего назначения, доступных пользователю. Сами регистры называются `%r0` - `%r31`, но у них есть логическое разделение:

Регистры общего назначения

Название	Имя внутри окна	Имя r-регистра
Глобальные (global)	%g0 - %g7	%r0 - %r7
Выходные (out)	%o0 - %o7	%r8 - %r15
Локальные (local)	%l0 - %l7	%r16 - %r23
Входные (in)	%i0 - %i7	%r24 - %r31

Данные регистры называются *r регистрами* и используются для целочисленных вычислений. Плавающие регистры называются *f регистрами*, они расположены отдельно, и о них мы сегодня говорить не будем. Интересно отметить, что сама архитектура предполагает от 64 до 528 *r регистров*, но регистровое окно содержит только 24. Чтение `%g0` всегда возвращает 0, а запись в него не даёт эффекта. Вообщем на спарке регистры сделаны очень круто, но их очень долго описывать, советую прочитать документацию [\[sparcv9\]](#).

Переходим к инструкциям. Начнём с инструкции `mov`. От интела эта инструкция отличается тем, что её нет в Спарке. Sparc - это RISC архитектура с малым количеством команд, но для удобства программистов ассемблер поддерживает синтетические инструкции. В частности приведённый `mov` возможно будет отранслирован следующим образом (есть несколько способов транслации в зависимости от аргументов):

```
mov 1, %o1->or %g0, 1, %o1
```

Синтетические инструкции не являются частью стандарта, но входят в информационное приложение к нему, так что их можно смело использовать.

Следующая инструкция `set`, являющаяся синонимом к инструкции `setuw`, которая тоже является синтетической инструкцией. Её раскрытие возможно выглядит следующим образом:

```
set hello_str %o2->
    or %o2, %lo(hello_str), %o2
```

Инструкция `sethi` поместит старшие 22 бита `hello_str` (т.е. её адрес) на регистр `%o2`.

Инструкция `or` поместит туда младший остаток. Обозначения `%hi` и `%lo` нужны для взятия старших и младших битов соответственно. Такие сложности возникают из-за того что инструкция кодируется 32 битами, и просто не может включать в себя 32-х битную константу.

Далее мы кладём значение 4 на глобальный регистр `%g1`. Можно догадаться что это номер вызова `write`. Системный вызов будет искать номер вызова именно там.

Операция `ta` инициирует системное прерывание. Её аргументом является тип системного прерывания. Скажу честно - я не нашёл нормального описания системных вызовов для v9, а то что туда надо подавать `0x10` выяснил случайно из архивов какой-то переписки. Поэтому придётся просто это запомнить :)

Далее производятся аналогичные действия для вызова `exit`, думаю их пояснить не нужно.

UPD:

Спасибо уважаемому Анониму за версию данной программы для SunOS 5.10:

```
.section ".text"
.global _start
_start:
    mov 4,%g1 ! 4 is SYS_write
    mov 1,%o0 ! 1 is stdout
    set .msg,%o1 ! pointer to buffer
    mov (.msgend-.msg),%o2 ! length
    ta 8

    mov 1,%g1 ! 1 is SYS_exit
    clr %o0 ! return status is 0
    ta 8

.msg:
.ascii "Hello world!\n"
.msgend:
```

Запуск:

```
$ as t1.s -o t1.o && ld t1.o && ./a.out
Hello world!
```

4. Эльбрус

Ну и, собственно, жемчужина коллекции - процессор Эльбрус. Работа проводилась на процессоре Эльбрус-4C, который имеет архитектуру команд v3 (наше внутреннее обозначение). Управляется машина ОС Эльбрус. Про сам Эльбрус можно почитать в [elbrus], про какую-либо документацию, находящуюся в открытом доступе мне неизвестно.

Как и Sparc, архитектура Эльбруса рассчитана в первую очередь на то что оптимальный код выдаст компилятор. Но в отличает от Sparc, ассемблер Эльбруса вообще не предназначен для людей. Итак, вот наш пример:

```
.section ".data"
$hello_msg:
.ascii "Hello, world\n\000"

.section ".text"
.global _start

_start:
! Подготавливаем вызов write
{
    sdisp %ctpr1, 0x3
    addd, 0 0x0, 13, %b[3]
    addd, 2 0x0, [_f64,_lts1 $hello_msg ], %b[2]
    addd, 1 0x0, 0x1, %b[1]
    addd, 3 0x0, 0x4, %b[0]
}

! Вызываем write
{
    call %ctpr1, wbs = 0x4
}

! Подготавливаем вызов exit
{
    sdisp %ctpr2, 0x1
    addd, 0 0x0, 0x0, %b[1]
    addd, 1 0x0, 0x1, %b[0]
}

! Вызываем exit
{
    call %ctpr2, wbs = 0x4
}
```

Сборка и запуск:

```
$ las t.s -o t.o && ld t.o && ./a.out
Hello, world
```

Начнём с изменения синтаксиса.

Мы видим что к синтаксису добавились фигурные скобки. Процессоры Эльбрус основаны на **VLIW** архитектуре, а значит могут исполнять множество статически спланированных команд за такт. Набор таких команд называется широкой командой (ШК) и заключается в фигурные скобки. Остальной синтаксис более или менее идентичен.

Если посмотреть на команду сборки, то вместо `as` используется `las`. Это наш местный ассемблер, но сейчас идёт процесс перехода на `gas`, поэтому скоро он станет неактуален (отдел, занимающийся ассемблером уже сейчас ругается если я его использую, но в дистрибутиве пока именно он).

Чтобы процессор мог исполнять много команд за такт, ему нужно много регистров. Согласен, что их никогда не бывает много, но для программы на Эльбрусе регистровый файл содержит 256 регистров общего назначения размером 64 бита. Из них 224 предназначены для процедурного стека, а 32 являются глобальными регистрами. В Эльбрусе нет отдельных регистров для плавающих вычислений, все они выполняются на одном конвейере и хранятся в общих регистрах. Именование регистров идёт следующим образом:

- `%r[<номер>]` - прямоадресуемые регистры текущего окна. `<номер>` является индексом относительно базы текущего окна
- `%b[<номер>]` - вращаемые регистры текущего окна. `<номер>` - индекс относительно текущей базы
- `%g[<номер>]` - глобальные регистры. `<номер>` является индексом относительно базы текущей глобальной области

Иногда в ассемблере регистры имеют различные префиксы. Подобные названия не влияют ни на что и нужны только для наглядности. Префиксы бывают следующие:

- `s` одинарный формат регистра - 32 бита (Single)
- `d` двойной формат регистра - 64 бита (Double)
- `x` расширенный двойной регистра - 80 бит (Extended)
- `q` квадро формат регистра - 128 бит (Quadro)

Существует программное соглашение, согласно которому для передачи аргументов в вызываемую процедуру мы используем вращающиеся регистры.

Итак теперь переходим к самой программе. Думаю первые несколько строк и так понятны, поэтому рассмотрим сразу первую ШК:

```
_start:  
{  
    sdisp %ctpr1, 0x3  
    addd, 0 0x0, 13, %b[3]  
    addd, 2 0x0, [ _f64, _lts1 $hello_msg ], %b[2]  
    addd, 1 0x0, 0x1, %b[1]  
    addd, 3 0x0, 0x4, %b[0]  
}
```

Рассмотрим первую команду `sdisp %ctpr1, 0x3`. А чтобы понять что это такое и что оно делает нужно ещё немного рассказать про механизм работы переходов в Эльбрусе. В процессорах Эльбруса вызов функции является дорогим удовольствием, поэтому переходы следует готовить заранее. Для этого существует два типа команд - `ctp` (подготовка перехода) и `ct` - фактический переход. Нам доступно три регистра перехода: `%ctpr1-%ctpr3`, т.е. за раз мы можем подготовить три маршрута для прыжка. Существует несколько команд подготовки перехода, нас здесь интересует `sdisp`. Эта команда подготавливает переход для системного вызова. Первым аргументом идёт регистр перехода, по которому мы будем совершать прыжок. Вторым аргументом - точка входа в операционную систему, нам она нужна равной 3 (64-х битный вход в ОС).

Далее рассмотрим команды `addd`. Как я уже говорил, ассемблер Эльбруса не предназначен для людей, и общепринятых мемоник здесь пока нет. Так в ассемблере нет команды `MOV`. Чтобы положить значение на регистр применяется команда `add`. Она производит сложение регистров или констант и записывает их в регистр.

Для Эльбруса одновременно доступно 6 арифметико-логических каналов (АЛК), т.е. за такт мы можем производить до 6 сложений. Итак, в первой операции мы кладём число 13 в регистр `%b[3]` - это длина нашей строки. (В версиях для других архитектур мы вычисляли это программно, и в Эльбрусе можно сделать также, но для `las` у меня это так и не получилось, хотя в `gas` всё заработало). Далее на регистр `%b[2]` мы кладём адрес начала нашего сообщения. Затем в `%b[1]` кладём идентификатор устройства вывода, и, наконец, в `%b[0]` кладём номер системного вызова. В целом аналогия с другими архитектурами прослеживается.

Далее может возникнуть вопрос зачем в команде `addd` третья `d`. В мемониках команд, реализованных для нескольких форматов операндов, последняя буква обозначает используемый формат. В данном случае мы работаем в `double` формате, т.е. с полноценным 64-х битным регистром.

Отдельно рассмотрим команду `addd, 2 0x0, [_f64, _lts1 $hello_msg], %b[2]`, которая, как можно догадаться, кладёт в регистр `%b[2]` адрес печатаемого сообщения. Для того чтобы кодировать адрес в памяти используется аргумент `[_f64, _lts1 $hello_msg]`. Квадратные скобки означают взятие адреса. Внутри расположен **длинный литерал**. Его содержимое означает следующее:

- `_f64` - формат литерала. В данном случае мы говорим что это литерал размера 64 (хотя он уместится и в 32 бита)
- `_lts1` - литеральный слог, кодирующий константное значение. Всего доступно 4 литеральных слога, так что в одной ШК мы не сможем поместить более 4 длинных литералов (в случае формата `_f64` - не более 2).
- `$hello_msg` - идентификатор, обозначающий нашу метку

Во второй ШК у нас производится операция `call %ctpr1, wbs = 0x4`, которая вызывает функцию, переход на которую подготовлен на регистре `%ctpr1`. т.е. вызывается наш `write`. Второй аргумент задаёт смещение для новой базы регистра окна. Здесь я не буду

объяснять что это значит, т.к. это займет много времени, просто пока придется запомнить что это должно быть так (на самом деле это очень частный случай и нужно понимать как его вычислять)

В третьей ШК мы аналогичным образом подготавливаем переходы для вызова `exit`, и в четвёртой ШК мы его вызываем.

Всё, проще некуда.

Послесловие

Как я уже говорил в начале, данный материал появился потому что я не смог найти чего-то подобного в сети. На самом деле многое я взял из этого [0xах] блога - описание примера на x86 и вообще саму идею. Для остальных архитектур пришлось изворачиваться :) Позже, во время работы над заметкой, я нашёл это [mechasm] неплохое описание, но оно уже было неактуально.

Вообще я планировал написать эту заметку за неделю-две и перейти на следующий пример. Более того хотел ещё включить описание llvm IR. Но внезапно простенькая заметка про hello world заняла у меня несколько месяцев. Преимущественно из-за Эльбруса. Тут оказалось много нового и непонятного при почти полном отсутствии читабельной документации. И тут хотелось бы сказать **огромное спасибо** многим коллегам, которые терпеливо в течении долгого времени разъясняли мне простейшие вещи.

В данной заметки могут быть неточности, ошибки и вообще фиг знает что, поэтому если что-то не так - пишите, я поправлю :)

Источники

[intel1] Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture
[intel2] Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B, 3C and 3D

[syscall1] Таблица системных вызовов linux

[syscall2] Другая таблица системных вызовов linux

[as] Мануал по ассемблеру

[0xах] Серия постов про написание hello world на ассемблере amd64. Во многом при написании заметки я смотрел именно в этот пост, там весьма подробное и доходчивое описание с замечаниями в комментариях

[mechasm] Аналогичный пост на русском, который я нашёл не сразу и не пользовался им. Но стиль изложения мне нравится

[sparcv9] The SPARC Architecture Manual Version 9

[sparcv9asm] SPARC Assembly Language Reference Manual

[oracle] Актуальная документация от Oracle

[sparcasmbook] SPARC Architecture, Assembly Language Programming, and C. Очень хороший учебник по ассемблеру и по спарку

[elbrus] Микропроцессоры и вычислительные комплексы семейства «Эльбрус»

Автор: alexanius на 19:50 40 Comments



Ярлыки: ассемблер, Эльбрус, sparc, x86

40 Комментариев

Войти



Присоединиться к обсуждению...

войти с помощью

или через DISQUS ?



Имя

Лучшее



alexanius • 6 лет назад

> Скажите, а можно объяснить постоянные теоретические рассуждения тем, что проект Эльбруса представлен как теоретический проект в основном в виде каких-то ссылок на результаты, полученные где-то.

Нельзя. Как минимум потому что утверждение "что проект Эльбруса представлен как теоретический проект" неверно и даже противоречит последней части фразы, т.к. теоретический проект не может давать непосредственно практических результатов. То что о нём мало информации в сети и мало у кого данная машина есть в наличии - да, хотя за последние пол года было несколько интересных заметок от людей, получивших данные машины в свои руки.

> А возможно ли такое, что если Эльбрус станет доступен практически, то количество теоретических рассуждений будет уменьшаться?

Он вполне доступен практически уже гораздо более широкому кругу лиц, чем, скажем, лет 5 назад. Пока это приводит к появлению редких, но вполне адекватных статей от пользователей. Тем не менее мне что-то подсказывает что при попадании машины к обычным пользователям количество даже не просто теоретических а хотя бы минимально

технических грамотных рассуждений будет составлять примерно нулевой процент из общего потока информации.

> Если Эльбрус изначально задуман, как проект для военных и для "узкого круга ограниченных людей", то зачем тогда о нем писать в технических новостях общего назначения?

Откуда взялось утверждение что он изначально задуман для военных?

> Конкретной ссылке в интернете доверять нельзя, оценку делать можно статистикой. Почему 80% ссылок на Эльбрус негативные, а только 20% - позитивные. В чем тут может быть проблема?

Хороший вопрос. Я думаю он из области психологии, но для начала можно уточнить как Вы получили эти цифры?

> И еще. А как можно утверждать, что статически можно сделать столько же, сколько и динамически? Вы действительно так считаете?

А можно процитировать высказывание, на основе которого Вы сделали данный вывод?

PS. Друзья, это статья про ассемблер, причём не только Эльбрусовский. Вопросы, не относящиеся к теме мешают людям, которые хотят почитать обсуждение данного примера ассемблера. Пожалуйста, найдите другой пост для вопросов общего характера про Эльбрус.

1 ^ | v • Ответить • Поделиться >



alexanius • 6 лет назад

И почему меня все пытаются убедить что Эльбрус - это плохо или что его вообще не существует о_о

> Почему VLIW это плохо.
> <http://thesz.livejournal.co...>

Сложности в создании VLIW систем обозначены, ок, во многом вполне справедливо. Но вот дальше начинается треш, а именно необоснованные предположения касательно производительности, подсчёты на пальцах и вообще цифры с потолка (это я про плотность кода).

> Теперь не просто про VLIW вообще, а про новый
> планируемый Эльбрус конкретно.
> <http://thesz.livejournal.co...>

Ну как будут серийные образцы 8С - так и посмотрим. Нет смысла гадать на кофейной гуще.

А вообще мне всё это читать очень забавно потому что пишут люди, которые не в теме. Не, конкретно товарищ thesz вроде довольно умный и даже в смежной области работает, но это всё теоретические рассуждения от человека, который не измерял реальную производительность Эльбрусов и не анализировал результаты.

Лично у меня нет оснований считать что VLIW системы - тупиковый путь развития, даже на десктопах. Ну т.е. да, на кривом софте оно будет работать медленнее но при повышенном частоте даже этого не будет заметно пользователю. А в перспективе VLIW систем вообще, и у Эльбруса в частности (там из киллер-фич не только VLIW если что) предоставиточно.

1 ^ | v • Ответить • Поделиться >



Сергей Зло ➔ alexanius • 6 лет назад • edited

А такой вопрос, чисто гипотетический. Предположим у нас кривой, но а точнее скажем софт x86 т.е. условно говоря он рассчитан 4-5 команд за такт, а не на 25. И условно говоря программа в среднем выполняет 4-5 команд за такт, максимум 10. Нельзя ли в этом случае сделать как бы гипертрейдинг (или это он и есть?) т.е. условно сказать что у нас не 4 ядра, а 8, но теперь они могут выполнять не 25 команд за такт, а максимум 12. Ну и регистров не 250, а 125 на ядро условно. Т.е. в этом случае же нагрузка на проц будет не 40%-50%, а 80-90, и таким образом, если сама прога поддерживает многоядерность, то можно ее ускорить в два раза почти. Понятно что там есть ограничения на АЛУ, т.е. получается не более 3 АЛ операций за такт и т.д., но в целом - насколько сложно это сделать?

^ | v • Ответить • Поделиться >



Nik Stasov • 3 года назад



Спасибо. Всё работает.
^ | v • Ответить • Поделиться >



alexanius • 6 лет назад
Как-то момент с 32-х битным кодом не заметил. А можно пример (исходник и ассемблер)? Пока не очень понятно о чём именно речь.

> Вообще такой режим врядли нужен.

Напротив, если программа в этом режиме падает, то это повод бить тревогу и искать ошибку в коде.

> В таком режиме весь open source будет валиться.

Будет валиться кривой код (ну иногда и не очень, но по большей части именно код, содержащий ошибки). Пишите письма разработчикам пакетов :)

> Такое тоже происходит на x86 если в llvm включить все возможные sanitizer-ы или взять valgrind.

Их как раз для этого и сделали. Кстати защищённый режим Эльбруса

[показать больше](#)

^ | v • Ответить • Поделиться >



Анонимный • 6 лет назад

Sorry, я неправильно спросил. А как получить 64-битовый код защищенного режима? С опцией -mptr128 он получается 32-битовым, а других опций нет?

Вообще такой режим врядли нужен. В таком режиме весь open source будет валиться. Любая aligned struct при копировании будет иметь неинициализированные байты в padding-ах. Такое тоже происходит на x86 если в llvm включить все возможные sanitizer-ы или взять valgrind.

E2k-машина - это же обычное дело, она же уже давным-давно выпускается серийно. :)

Sorry, никаких блогов я не веду. Если чесно, то мне вообще страшно касаться тем с засекреченной системой команд, я даже выхожу через какие-то proxy-сервера в Европе на всякий случай. Кто знает, а вдруг при диктатуре дадут команду 66 убрать всех кто слышал что-либо краем уха. :)

^ | v • Ответить • Поделиться >



alexanius • 6 лет назад

Блин, блоггеровские комментарии не поддерживают тэг code. Надо что-то нормальное для комментов прикручивать...

^ | v • Ответить • Поделиться >



alexanius • 6 лет назад

Ооо, -mptr128 - это тот самый защищённый режим - одна из киллер-фич. Подробные материалы есть на сайте МЦСТ, но если вкратце, то он позволяет падать если мы сделали например выход за границу массива или работаем с неинициализированными значениями. И всё это поддерживается аппаратно.

Например:

```
$ cat t.c
#include
```

```
int main()
{
    int i;
```

```
    printf("%d\n", i);
```

```
    return 0;
}
```

[показать больше](#)

^ | v • Ответить • Поделиться >



Анонимный • 6 лет назад

Спасибо. У меня на Итаниуме можно загружаться под HP-UX и компилировать родным aCC от HP. Оптимизирует он лучше чем gcc по скорости. Но и gcc и aCC раздувают код своими пропатчами почти

акрости Phi и Phi, и где разделяют код, скомпилированный на Phi
одинаково. Интеловский по-моему нужен только для native Phi (icc -mmic).
Вопрос. А что такое компиляция в защищенном режиме (icc -mprt128).
Получается 32-битный код, что это еще такое?

^ | v • Ответить • Поделиться >



alexanius • 6 лет назад

Ваш комментарий меня крайне улыбнул :) Но про политику лучше не надо :) Давайте на Ваш вопрос про информационный фон я отвечу чуть попозже и, пожалуй, создам для этого отдельный пост, а сейчас отвечу на техническую часть.

> Получил ассемблерный код для обеих машин. Как можно посчитать, где плотность кода лучше?

Короткий ответ - не знаю.

Длинный - немного распишу. Для начала нужно определить что понимать под "плотностью кода". Я вижу это как среднее количество реально исполненных инструкций за такт - IPC. Для x86 есть утилита perf, которая умеет данный параметр отображать (кстати очень советую на неё взглянуть людям, утверждающим что intel умеет 4 или уже 8 инструкций за такт). В Эльбрусе данная утилита есть, но конкретно эта функциональность на сколько мне известно не работает. Это связано со сложностями работы с широкой командой. Какое её состояние в Итаниуме

показать больше

^ | v • Ответить • Поделиться >



Анонимный • 6 лет назад

Да, есть много других политических блогов, на которых ничего не узнаешь. Пишу тут, потому что хочется узнать из первых рук мнение профи, кто пишет на ассемблере под эльбрус. Давайте, чтобы было все в порядке, я буду спрашивать общий вопрос и сразу что-то по ассемблеру. (Если так плохо, то напишите, чтобы я больше не писал вообще, и я больше писать не буду).

Однажды к Путину на передачу Ярмольник привел Шевчука из ДДТ. Шевчук спросил Путина, почему все так плохо. В ответ получил, что не надо все уравнивать, были отдельные частные хорошие случаи, про которые не надо забывать, и начал их перечислять в качестве ответа. У Вас ответы очень похожие - на вопрос о тенденции, Вы отвечаете перечислениями частных случаев, которые не меняют этой тенденции. (80% случаев - это тенденция :)

Вот я взял у себя для сравнения одну машину с эльбрусом и одну с Итаниумом (обе VLIW).

debianle2k \$ icc -v

показать больше

^ | v • Ответить • Поделиться >



Анонимный • 6 лет назад

Скажите, а можно объяснить постоянные теоретические рассуждения тем, что проект Эльбруса представлен как теоретический проект в основном в виде каких-то ссылок на результаты, полученные где-то. А возможно ли такое, что если Эльбрус станет доступен практически, то количество теоретических рассуждений будет уменьшаться?

Если Эльбрус изначально задуман, как проект для военных и для "узкого круга ограниченных людей", то зачем тогда о нем писать в технических новостях общего назначения?

Конкретной ссылке в интернете доверять нельзя, оценку делать можно статистикой. Почему 80% ссылок на Эльбрус негативные, а только 20% - позитивные. В чем тут может быть проблема?

И еще. А как можно утверждать, что статически можно сделать столько же, сколько и динамически? Вы действительно так считаете?

^ | v • Ответить • Поделиться >



Анонимный • 6 лет назад

Почему VLIW это плохо.

<http://thesz.livejournal.co...>

Теперь не просто про VLIW вообще, а про новый планируемый Эльбрус конкретно.

<http://thesz.livejournal.co...>

^ | v • Ответить • Поделиться >

• | • Ответить • Поделиться >



alexanius • 6 лет назад

> Очередная вершина Эльбруса. Виртуозам ПиАра посвящается

Спасибо, поржал. Обожаю пропагандистские статьи, пишущиеся людьми, не разбирающимися в предмете :)

• | • Ответить • Поделиться >



alexanius • 6 лет назад

> "Нет. Ровно столько же." - не может быть такого. Это же противоречит нашему "IA64 не смог победить".

Не вижу противоречия. А под динамикой Вы вообще что понимаете? Out-Of-Order или динамические языки? В обоих случаях множество решаемых задач одинаково.

> Вложенные if-ы будет трудно запретить в демократическом обществе...

Если даже goto до конца не добили... Но кому нужна скорость - быстро от них откажутся.

> Бабаян - это имя нарицательное.

Ну оно как бы должно что-то означать. Я не понимаю что под этим подразумевается и причём он вообще здесь)

> "движение в сторону увеличения количества ядер" не имеет ничего

показать больше

• | • Ответить • Поделиться >



alexanius • 6 лет назад

Вы вырвали какой-то кусок обсуждения из непойми откуда (не, я знаю что с ЛОРа, но всё же), и хотите у меня узнать что имел ввиду его автор? :)

Так вот: не знаю, спросите у автора. Я не очень понял про какое взаимодействие идёт речь. Если зададите конкретный вопрос - постараюсь ответить (про JVM - вообще мало что могу рассказать, но недавно на явовской конференции [доклад](#) был про портирование).

• | • Ответить • Поделиться >



Анонимный • 6 лет назад

А прокомментируйте пожалуйста вот это:

> Каким образом, в рантайме, «Эльбрус» будет собирать потоки из различных, не взаимосвязанных, процессов ?

> > Не знаю, а как сейчас работает?

> А сейчас оно не работает, на каждый процесс свой контекст.

Что здесь имеется в виду? Что в ОС или скажем в JVM на эльбрусе нельзя обеспечить взаимодействие между одновременно работающими приложениями (как я понимаю) и на сколько это соответствует действительности?

• | • Ответить • Поделиться >



alexanius • 6 лет назад

> Приятно посмотреть, что Эльбрус выполняет сразу столько операций сложений. А почему в SPARC и x86 эти операции совсем не нужны?

Потому что в Эльбрусовской системе команд отсутствует инструкция MOV (как и в Спарковской, между прочим). Т.е. здесь addd выполняет роль MOV, только и всего.

• | • Ответить • Поделиться >



alexanius • 6 лет назад

> Интересно, а что напечатает Эльбрус после того как его компилятор проанализирует все зависимости данных?

Вы же понимаете, что программа содержит UB и результат может быть абсолютно произвольным?

• | • Ответить • Поделиться >



Анонимный • 6 лет назад

Интересно, а что напечатает Эльбрус после того как его компилятор проанализирует все зависимости данных?

int a;

a=1;

```
a=1;
a=a+(a=10);
printf("a=%d\n", a);
a=1;
a=a*2+(a=10);
printf("a=%d", a);
printf("a=%d, a=%d\n", a=2, a*2);
^ | v • Ответить • Поделиться >
```



Анонимный • 6 лет назад

Приятно посмотреть, что Эльбрус выполняет сразу столько операций сложений. А почему в SPARC и x86 эти операции совсем не нужны?

^ | v • Ответить • Поделиться >



alexanius • 6 лет назад

Спасибо, добавил к посту

^ | v • Ответить • Поделиться >



Анонимный • 6 лет назад

```
$ ed
a
.section ".text"
.global _start
_start:
    mov 4,%g1 ! 4 is SYS_write
    mov 1,%o0 ! 1 is stdout
    set .msg,%o1 ! pointer to buffer
    mov (.msgend-.msg),%o2 ! length
    ta 8

    mov 1,%g1 ! 1 is SYS_exit
    clr %o0 ! return status is 0
    ta 8

.msg:
.ascii "Hello world!\n"
.msgend:
.
w t1.s
283
q

$ as t1.s -o t1.o && ld t1.o && ./a.out
Hello world!
```

^ | v • Ответить • Поделиться >



Анонимный • 6 лет назад

"Нет. Ровно столько же." - не может быть такого. Это же противоречит нашему "IA64 не смог победить".

Вложенные if-ы будет трудно запретить в демократическом обществе - будут думать: это Хрущев опять учит художников, что нужно рисовать. :)

Бабаян - это имя нарицательное. Например, страна может еще долго оставаться ленинской даже после ухода Ленина. Как за 10 лет может смениться 3 поколения? :)

"движение в сторону увеличения количества ядер" не имеет ничего общего с улучшениями заполнения VLIW. Ваш компилятор имеет road map? Он будет распараллеливать по ядрам статически или динамически на виртуальной машине потока данных (что-то как в TERAFLUX, TALM/Trebuchet, BMDFM, etc.)?

^ | v • Ответить • Поделиться >



alexanius • 6 лет назад

> Динамически можно делать больше, чем статически.

Нет. Ровно столько же.

> Даже если прятать if-ы в предикатные инструкции, то что делать, если есть много вложенных if-ов и т.д. и т.п...

Страдать. Причём независимо от архитектуры. Много ветвлений в горячем коде - очень тревожный звоночек и скорей всего люди, которые писали данный код просто не думали о производительности.

Вообще мы имеем такую ситуацию с языками программирования и софтом

отчасти потому что в свое время IA64 не смог победить. Но код часто можно переписать по-человечески. Можно и не переписывать, просто работать будет чуть медленнее.

> Скажите как эксперт, вращающийся в Бабаяновских кругах

[показать больше](#)

^ | v • Ответить • Поделиться >



alexianus • 6 лет назад

\$ uname -srpv
SunOS 5.9 Generic_112233-11 sparc

```
$ as --version
GNU assembler 2.11.2
Copyright 2001 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License. This program has absolutely no warranty.
This assembler was configured for a target of `sparc-sun-solaris2.8'.
```

\$ as t.s -o t.o

\$ ld t.o -o a.out

```
$ ./a.out
Illegal Instruction (core dumped)
```

Рассмотрим проблему, возникшую у Вас. Видно, что Вы используете as от

[показать больше](#)

^ | v • Ответить • Поделиться >



Анонимный • 6 лет назад

\$ uname -srpv
SunOS 5.10 Generic_137137-09 sparc

```
$ as -V
as: SunOS 5.10 118683-10 Patch 03/14/2013
```

```
$ ed
a
.section .data
hello_str:
.ascii "Hello, world\n"
.set hello_str_len, . - hello_str
```

.global _start

.section .text

start:

[показать больше](#)

^ | v • Ответить • Поделиться >



Анонимный • 6 лет назад

Идея пар-ядер big.LITTLE это от бедности, а не от изобретательности. То что канаёт для карманных вычислителей уже не канаёт для больших и мощных CPU, тут можно только переизобретать P/S/C-State's

^ | v • Ответить • Поделиться >



Анонимный • 6 лет назад

Мы все сидим под Qtcreator-ом. У нас на одном Хеон-сервере под ESX-виртуализацией сидит 125 человек под своими Унихами с X-сами и каждый под своим Qtcreator-ом в VNC-сессии - не в этом дело. Динамически можно делать больше, чем статически. Даже если прятать if-ы в предикатные инструкции, то что делать, если есть много вложенных if-ов и т.д. и т.п...

Что сделали на ARM-е. Каждое ведро имеет два подведра. Одно подведро - полный dynamic multi-issue superscalar, а другое - простое, как двери. При сильной загрузке работает первое подведро, а при низкой - второе с целью energy-safe.

Скажите как эксперт, вращающийся в Бабаяновских кругах, а Бабаяновский-team когда-либо рассматривал следующую идею эльбруса (Эльбрус с большой буквы для меня звучит как гора, что-то вроде земля и Земля как планета): не напихивать die чисто Бабаяновскими VI IW-вядрами, а

напомню, что на момент написания этого впечатления VLIW вдребезги, а сделать каждое ведро из двух подведер: полного традиционного dynamic multi-issue superscalar и чисто Бабаяновского. Рафинированные подходы нигде не работают идеально, зато отлично работают гибридные. Пусть даже их instruction set будет уникальнейший отечественный, если того так требует импортозамещение, но 80% кода, которым компилятор не смог заполнить VLIW (ну не смог, и ладненько), отработает на dynamic multi-issue superscalar-e, а 20% кода будет помечено как "удачно заполненный VLIW" отработает на Бабаяновском ведре. Все будут щасливы, и правило 20/80 тоже там где надо. :)

^ | v • Ответить • Поделиться >



alexanius • 6 лет назад

Довольно странные допущения, причём вообще все. Если Вас интересуют какие-то реальные тесты, то вот отличная серия статей с хабра про тестирование Эльбрусов и потом наши комментарии, которые мы дали тем ребятам после публикации их результатов.

Если рассматривать вариант с подкупом и встраиванием MUX, то это нужно будет подкупать кого-нибудь и сотрудников МЦСТ, что вообще мягко говоря сомнительно :) А за то что их поставят на фабрике я бы не особо беспокоился.

Я не сказал бы что VLIW бессилен на динамических языках, но да, определённые проблемы есть. Не знаю, получится ли сделать их работу быстрее чем на x86, но сделать их скорость сопоставимой можно.

Да, забить широкую команду сложно, особенно если идёт программа с большим количеством ветвлений. Это прям антипаттерн для нас. Тем не менее есть способы решения этих проблем. Например есть предикатный режим, есть спекулятивное исполнение. В конце концов есть if-conversion,

показать больше

^ | v • Ответить • Поделиться >



Анонимный • 6 лет назад

Возможно это все и существует, но не в состоянии military quality.

Процесс закладывания закладки может происходить приблизительно так: платят большие деньги кому-то, кто на каком-то этапе подготовки layout-a добавляет простенький MUX подключенный к шине команд и реагирующий на VLIW="010101...010101". По приходе такого длинного слова MUX отрубает что-нибудь. Человек, который это сделал пропадает с панамским паспортом в кармане. :)

Идея эльбруса такая: без формальных доказательств предположили, что multi-issue superscalar не сможет выполнять больше 6 инструкций параллельно, взяли и выбросили из процессора ту часть, которая пытается динамически накормить все ALU - ломать не строить; потом начали себя расхваливать, какие мы молодцы, пускай все себе делают сложные процессоры, а мы из нашего процессора все выбросили - наш компилятор и двоичный транслятор, получающий доступ к состоянию процессора, сделает всю работу еще лучше (опять без формальных доказательств); потом начали себя расхваливать и за это тоже. А

показать больше

^ | v • Ответить • Поделиться >



alexanius • 6 лет назад

Не очень понял что Вы имели ввиду, но могу сказать что во-первых Эльбрусы не производятся в Китае, а во-вторых я бы очень хотел посмотреть на процесс добавления закладки в процессор на этапе напыления транзисторов)

^ | v • Ответить • Поделиться >



alexanius • 6 лет назад

Сейчас нет, но очень может быть что вскоре ситуация поменяется.

А Вы, простите, в чём именно сомневаетесь? В существовании кросового компилятора, существовании Эльбрус ОС или существовании процессора Эльбрус?))

^ | v • Ответить • Поделиться >



Анонимный • 6 лет назад

Начал писать на mcst@mcst.ru, а потом подумал и остановился. Эльбрус поставлен на защиту кумовского капитализма от всего остального мира. Не дай Бог китайцы на фабрике засунут закладку в эльбрус, тогда кумовской капитализм отрежет головы даже тем, кто знал как написать "hello world" на

асме. :)

[^](#) | [v](#) • Ответить • Поделиться >



Анонимный • 6 лет назад

Спасибо. А можно где-то получить cross toolchain для эльбруса или qemu image эльбруса?

Очень похоже на сюжет фильма Terminator 3, где герои приходят к выводу, что core of Skynet does not exist at all. :)

[^](#) | [v](#) • Ответить • Поделиться >



alexanius • 6 лет назад

Получение удалённого логина - штука довольно неофициальная. Можете попробовать написать на адрес mcst@mcst.ru. Сразу дам совет - представьтесь, напишите какие тесты хотите запускать, куда пойдут результаты. Тогда Вам может быть даже ответят.

[^](#) | [v](#) • Ответить • Поделиться >



Анонимный • 6 лет назад

А где можно получить удаленный логин на эльбрус?

[^](#) | [v](#) • Ответить • Поделиться >



alexanius • 6 лет назад

Мы никакую процедуру нигде не описывали. И ассемблер всё же не совсем обычный язык. Суть в том что для того чтобы переключить исполнение на другую функцию нужны определённые затраты. В случае с Эльбрусом, процессор не умеет предсказывать куда мы заранее пойдём, поэтому мы готовим переходы заранее (в данном случае инструкцией sdisp). Т.о. получается что сам sdisp позволяет процессору подгрузить заранее инструкции на исполнение, а call вообще не будет затрачивать времени на переход. Если посмотреть в ассемблер intel или sparc, то там такой инструкции нет.

[^](#) | [v](#) • Ответить • Поделиться >



Анонимный • 6 лет назад

Круто выглядит, синтаксис практически явущийся.

> нужно ещё немного рассказать про механизм работы переходов в Эльбрусе. В процессорах Эльбрус вызов функции является дорогим удовольствием

А в чём проблема то, ну описали в начале процедуру/функцию, ну вызвали - ровно точно такой же подход разве не в любом (практически) современном языке?

[^](#) | [v](#) • Ответить • Поделиться >



Анонимный • 6 лет назад

Очередная вершина Эльбруса. Виртуозам ПиАра посвящается

<http://worldcrisis.ru/crisis...>

[^](#) | [v](#) 1 • Ответить • Поделиться >



Подписаться



Privacy



Не продавайте мои данные

DISQUS

[Следующее](#)

[Главная страница](#)

[Предыдущее](#)

Подписаться на: Комментарии к сообщению (Atom)

Тема "Корпорация "Чудеса"". Технологии Blogger.