Архитектура процессоров Мультиклет

М.О. Бахтерев

Институт Математики и Механики им. Н.Н. Красовского УрО РАН,
Екатеринбург

RuCTF-2015

Математика традиционных машин

$$S_{t+1} = I \cdot S_t$$

- I – оператор, задаваемый очередной инструкцией.
- S_t – состояние системы в момент t.
- На самом деле, это фактически монада.

Некоторые нетривиальности $S_{t+1} = I \cdot S_t$

- $S_t = S{:}CPU_t \times S{:}RAM_t$; но в реальной системе много динамических процессов: состояний-произведений (product states из QM) может и не быть.
- t – ещё одна неочевидная сущность. Нужны часы Лампорта, похожие на время ОТО, а не привычное линейное время.
- Математики взаимодействий пока нет (насколько мне известно), поэтому не рассматриваем `I/O`.
  - Ближе всего к решению вопроса интерпретации КМ: математика запутанных состояний позволяет «непротиворечиво» разделять системы на внешний и внутренний мир.
  - В информатике эта проблематика возникает в распределённых системах. Математика:
    - CSP;
    - П-исчисление;
    - RiDE.

Основное узкое место $S_{t+n} = I_n \cdot I_{n-1} \cdots I_1 \cdot S_t$

- На кристалле можно разместить очень много ФУ:
  - даже в среднем GPU их больше 500.
- Задержки, однако

```
int fn(Some *st, double x, double y) {
  st->some = x/y;
  st->other = st->z + x*y;
  return x-y; }
```

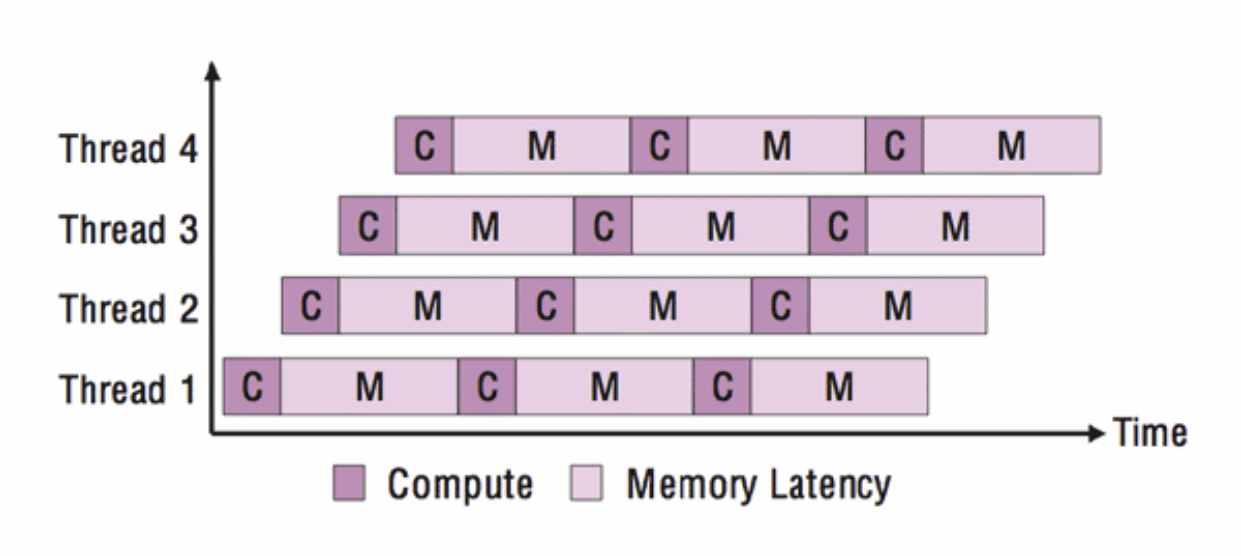
- Последовательное исполнение (in-order-execution): пока не завершена I_k к выполнению I_{k+1} не приступаем:
  - Вычисление `st->other` не начнётся до завершения `x/z`.
  - Вычисление `x-y` не начнётся до завершения записи в `st->other`.

Деления и доступы в память – операции очень долгие.

Можно ли быстрее?

CMT – быстрее $S_{t+n} = I_n \cdot I_{n-1} \cdots I_1 \cdot S_t$

Инструкции есть в другой нити: $S{:}CPU^k_{t+n} = I^k_n \cdots I^k_1 \cdot S{:}CPU^k_t$

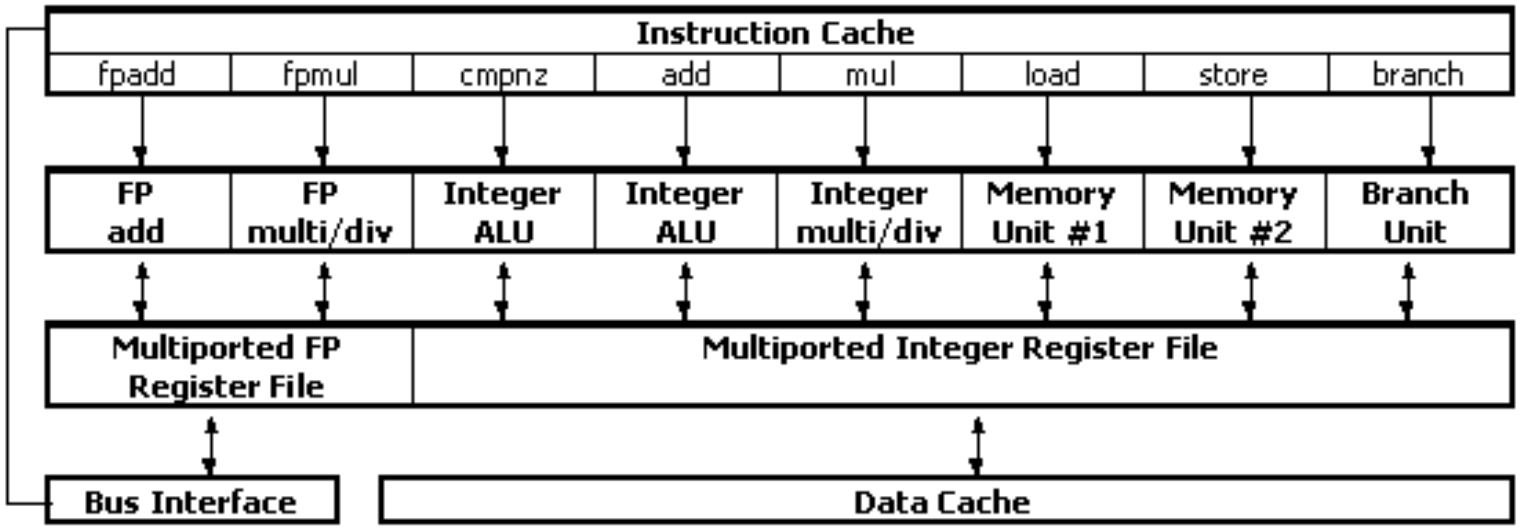


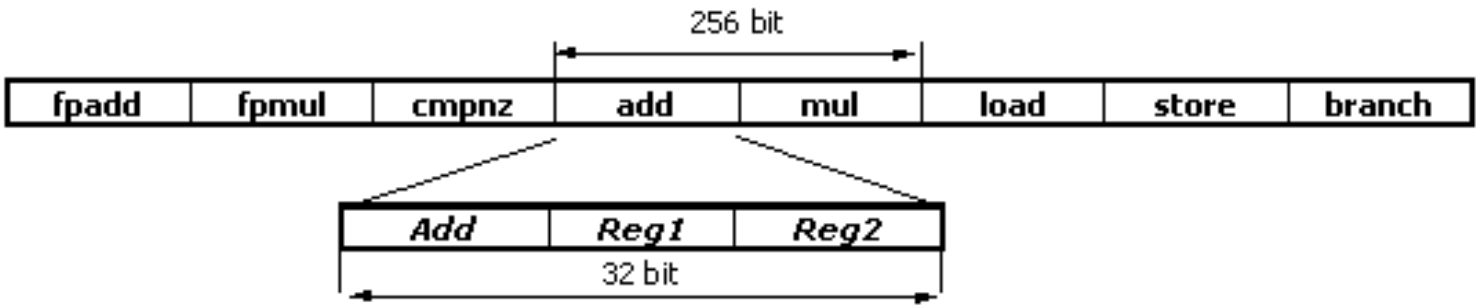
Недорого, энергоэффективно, но требует разбиения на нити и организации их взаимодействия.

- Niagara;
- PowerPC A2 – процессоры BlueGene/Q и XBox 360;
- составляющая эффективности GPU.

VLIW – быстрее $S_{t+n} = I_n \cdot I_{n-1} \cdots I_1 \cdot S_t$

Можно сделать инструкции сложными: $I_k = i_k^0 \ldots i_k^L$





VLIW – быстрее $S_{t+n} = I_n \cdot I_{n-1} \cdots I_1 \cdot S_t$

Допустим, $I_k = i_k^0 \ldots i_k^L$.

- Эффективный компилятор вполне можно разработать, но остаются проблемы.
- Если i_k^M – приводит к долгой записи в память или делению, нужно ждать завершения.
- EPIC-чески сложное решение в Intel Itanium – программно-аппаратная реализация операций:
  - промах по кэшу $\rightarrow$ прерывание $\rightarrow$ программная предвыборка данных;
  - деление итерациями метода Ньютона, при нехватке точности $\rightarrow$ прерывание $\rightarrow$ программное деление столбиком.
- Однако, решение популярно, в системах с известными задержками:
  - Эльбрус 2k;
  - TMS3206x;
  - TILE64.

OoOE – быстрый $S_{t+n} = I_n \cdot I_{n-1} \cdots I_1 \cdot S_t$

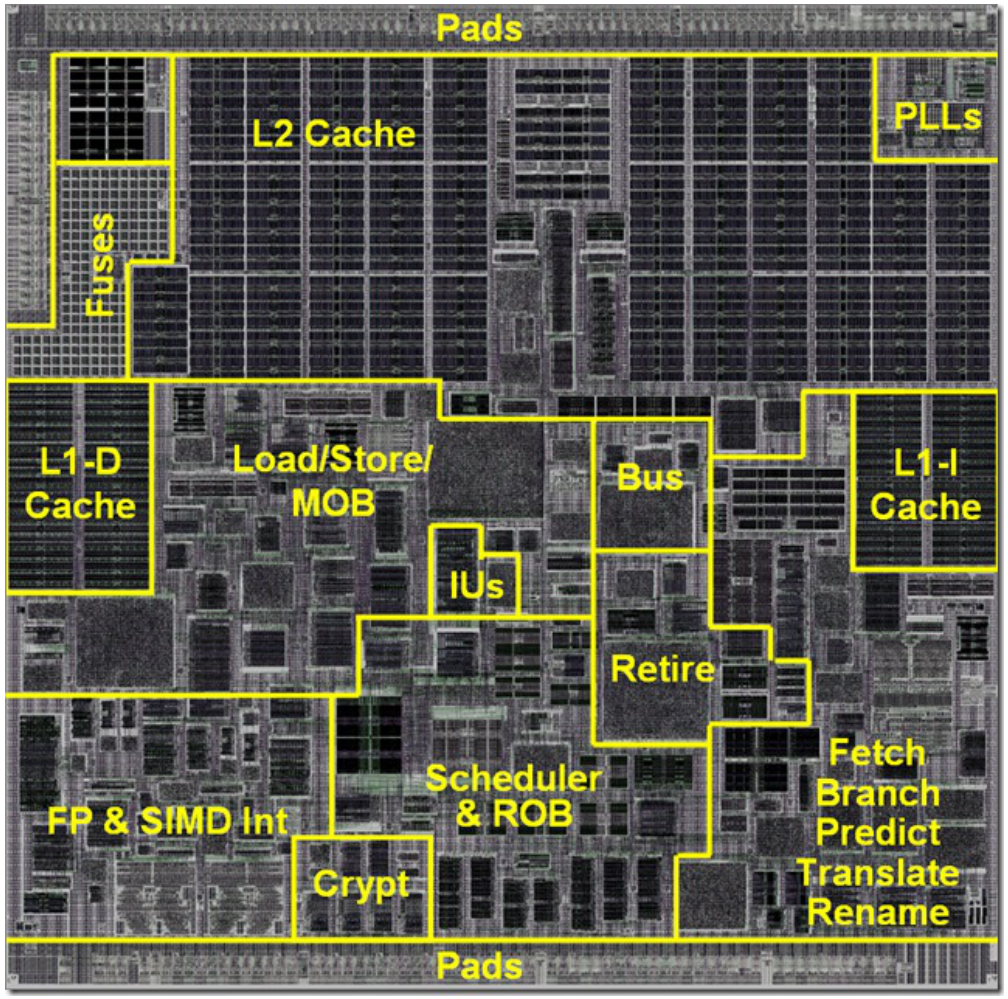
Можно выполнять инструкции в порядке (частичном) их зависимости. Для этого необходимо:

- переименовать регистры, для выявления параллелизма

```
add rax, rbx       add r48 = r90, r56
mov [rax], 1   →   mov [r48] = 1
mul rax, rcx       mul r42 = r48, r18
```

- поддерживать очереди планирования по готовности;
- разобраться с очерёдностью доступа в память;
- подтверждать выполнение инструкций с учётом прерываний и исходного порядка: если выполнение `mov [r48] = 1` вызвало исключение, то архитектурный `rax` не следует связывать с `r48`;
- предсказывать переходы для загрузки оборудования.

Цена технологии Out-of-Order Execution



Возможно ли более экономное решение?

Ассемблер МСр. Параграф

```
 1  fn5.P3:
 2    jmp   fn5.P4
 3    rdl   fn5.3.44RT
 4    addl  @1, 0x00000011
 5    rdsb  @1
 6    exa   fn5.a.24A + 4
 7    addl  @2, @1
 8    rdsb  @1
 9    rdsl  fn5.1.36RT
10    mulsl @1, @2
11    wrsl  @1, fn5.4.48RT
12    complete
```

Ассемблер MCp. Параграфы

```
 1  fn5.P5.blkloop:
 2    exa  #CX
 3    jne  @1, fn5.P5.blkloop
 4    je   @2, fn5.PF
 5    rdb  #SI
 6    wrb  @1, #DI
 7    irm  0x38
 8    complete
 9
10  fn5.PF:
11    rdl   #BP, 4
12    jmp   @1
13    getl  #BP
14    rdl   #BP, 0
15    addl  @2, 4
16    setl  #BP, @2
17    setl  #SP, @2
18    complete
```

Исполнение параграфов

4 клетки

```
fn5.P5.blkloop:
  exa  #CX                    ; PU0
  jne  @1, fn5.P5.blkloop     ; PU1
  je   @2, fn5.PF             ; PU2
  rdb  #SI                    ; PU3
  wrb  @1, #DI                ; PU0
  irm  0x38                   ; PU1
  complete                    ; PU1

fn5.PF:
  rdl   #BP, 4                ; PU0
  jmp   @1                    ; PU1
  getl  #BP                   ; PU2
  rdl   #BP, 0                ; PU3
  addl  @2, 4                 ; PU0
  setl  #BP, @2               ; PU1
  setl  #SP, @2               ; PU2
  complete                    ; PU2
```

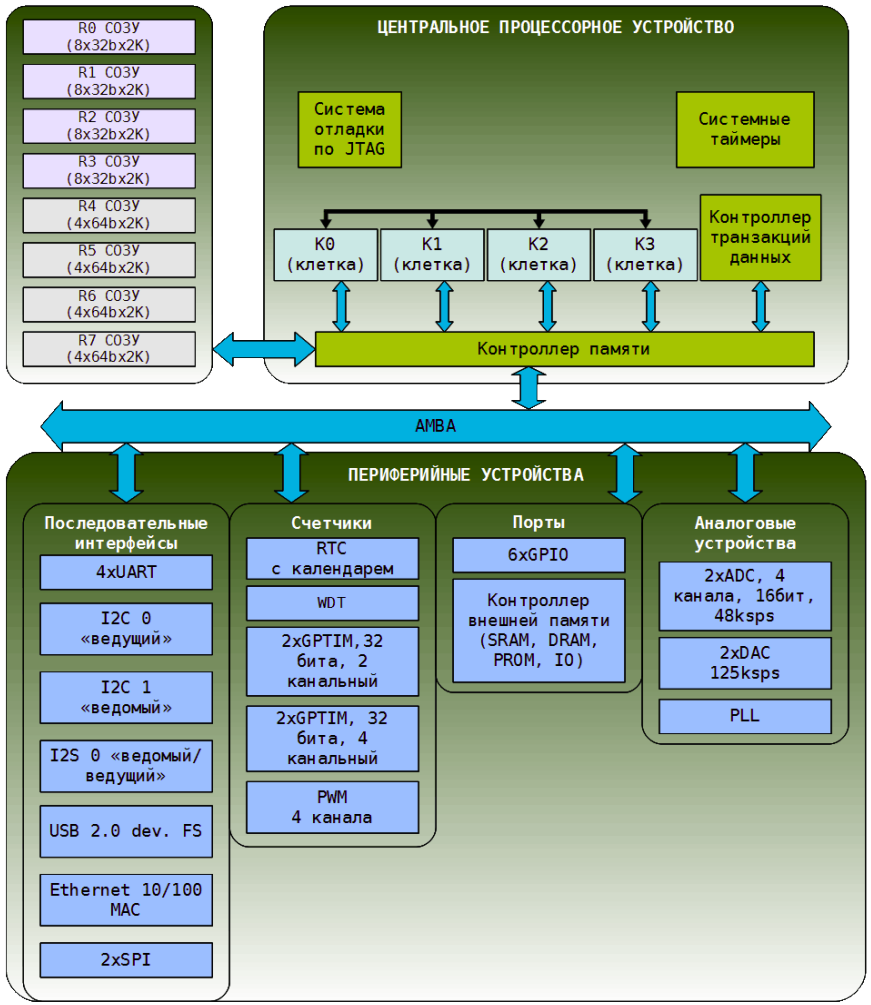
2 клетки (i.e., остальные defunct)

```
fn5.P5.blkloop:
  exa  #CX                    ; PU0
  jne  @1, fn5.P5.blkloop     ; PU1
  je   @2, fn5.PF             ; PU0
  rdb  #SI                    ; PU1
  wrb  @1, #DI                ; PU0
  irm  0x38                   ; PU1
  complete

fn5.PF:
  rdl   #BP, 4                ; PU0
  jmp   @1                    ; PU1
  getl  #BP                   ; PU0
  rdl   #BP, 0                ; PU1
  addl  @2, 4                 ; PU0
  setl  #BP, @2               ; PU1
  setl  #SP, @2               ; PU0
  complete                    ; PU0
```

Это всё в железе MCp042R100102 LQ256

Это всё в железе MCp042R100102 LQ256



С точки зрения нашей «микроматематики»

Каждый оператор I, меняющий состояние $S_{t+1} = I \cdot S_t$, задаётся цепочкой связанных друг с другом инструкций:

$$I = i_0 i_1 \ldots i_{k_I}.$$

Архитектура, несомненно, попадает в класс EPIC с явным исполнением графа программы. Но:

- это не VLIW, потому что инструкции не связаны жёстко с ФУ и мультипортовым регистровым файлом, они связаны друг с другом;
- это не EDGE (с которым часто путают MCp в «интернетах»), потому что инструкции не привязаны к аппаратным очередям данных, они оперируют самими данными.

Profit. Экономия транзисторов и энергии

- Нет необходимости в сложных устройствах переименования регистров. Программа и без того в dataflow-виде.
- Можно точно программировать переходы параллельно с исполнением кода, нет необходимости их предсказывать.
- Нет необходимости заниматься вычислением порядка операций с памятью, они упорядочены «естественно»: кодом программы и флагами `complete`.
- Не нужно заниматься подтверждением и отставкой инструкций. Модель ISA для программиста такова, что они могут выполняться в порядке (частичном), задаваемом их зависимостями.
- При всём этом производительность не опускается ниже уровня современных OoOE-процессоров (бывает и выше).

Profit. Новые возможности

- Ещё один уровень защиты ответственных систем от сбоев: вышедшие из строя клетки можно отключать, сохраняя работоспособность системы.
- Можно настраивать систему под особенности алгоритма:
  - собрать 4 клетки в один 4-way суперскалярный CPU;
  - разбить их на 4 независимых процессора, подобных ФУ GPU, и обрабатывать 4 нити параллельно;
  - от 1 до 4 контроллеров по «аппаратной» цене одного.
- Новые занятные возможности в реализации языков программирования высокого уровня:
  - хорошо раскладываются вычисления чистых функций;
  - способ программирования переходов хорошо подходит для безопасных языков.
- Энергоэффективность. Вместо `big.LITTLE` просто отключаем «лишние» клетки, без затрат на перенос нитей на другие ядра и разогрев кэшей.

Инструменты для разработки на Си-89

- `mc-as` – ассемблер с вполне традиционными возможностями. Можно не считать ссылки на коммутатор, а именовать результаты инструкций:

```
a1l  :=  rdl  mc_POPCNT64.ARG1l
a1h  :=  rdl  mc_POPCNT64.ARG1h
a1ll :=  and  @a1l, 0xFFFF
a1hl :=  and  @a1h, 0xFFFF
a1lh :=  slrl @a1l, 0x10
a1hh :=  slrl @a1h, 0x10
```

- `mc-ld` – редактор связей. Вполне стандартно используется для подготовки загрузочного образа отладочной платы:

```
mc-as some-asm.s -o some.o \
  && mc-as other-asm.s -o other.o \
  && mc-ld some.o other.o -o image.img
```

Инструменты для разработки на Си-89

- `mc-ploader` – загружает образ на системную плату

```
mc-ploader image.img
```

- `mc-rcc` – компилятор ANSI C на основе компилятора LCC. При запуске следует указать нужный целевой процессор:

```
mc-rcc -target=mcp \
  < preprocessed-source.i \
  > assembly.s
```

- `mc-mcpp` – стандартный сторонний препроцессор:

```
mc-mcpp -I ${MCP_SDK_ROOT}/include \
  < source.c \
  > preprocessed-source.i
```

Инструменты для разработки на Си-89

- `mc-lcc` – драйвер to rule them all:

```
mc-lcc -lccdir ${MCP_SDK_ROOT} -o image.img \
  some.a \
  other.c \
  compiled.o
```

- `mc-dbg` – отладчик, напоминающий gdb. Позволяет работать в режиме эмуляции процессора и собирать детальную трассу программы

```
{ echo -e 'm a sw\nr'
  sleep 10s
  echo -e 'm dump mem.txt\nq' } \
  | mc-dbg image.img > log.txt
```

Инструменты для разработки на Си-89

- `tools/tcp_client` – для обмена с моделью, запущенной `mc-dbg`, через «виртуальные» UART

Полезные ссылки

Руководство пользователя

- http://multiclet.com/docs/PO/Manual_Soft.pdf

Компоненты SDK

- Windows
  http://multiclet.com/docs/PO/MultiCletSDK_for_P1_ru.20141211.exe
- Linux
  http://multiclet.com/docs/PO/MultiCletSDK_for_P1.20141211.tar.gz

Полезные ссылки

Различные ресурсы

- Официально
  `http://multiclet.com/index.php/ru/support/technical-documents`
- Примеры
  `http://multiclet.com/community/projects/examples/files`

Спасибо за внимание

«Улыбаемся и машем, ребята, улыбаемся и машем»

```
#include <HDL51001_ccf.h>
#include <HDL50001_gpio.h>

void main() {
 while(1) {
  int i;
  GPIOB->DIR = ((uint32_t)0x60000000);
  GPIOB->OUT = ((uint32_t)0x60000000);
  for(i=0; i<1000000; i++);
  GPIOB->DIR = ((uint32_t)0x60000000);
  GPIOB->OUT = ((uint32_t)0x00000000);
  for(i=0; i<1000000; i++);
 }
}
```

http://k.imm.uran.ru/tmp/mcp-ructf-2015.pdf