1 Архитектура компьютера MIPT

MIPT — машина с архитектурой Фон-Неймана с адресным пространством в 2^{20} слов, каждое из которых занимает 32 бита.

Каждая команда занимает ровно одно слово, 8 старших бит которого — код операции, а использование остальных 24 битов зависит от операции.

Компьютер оснащён шестнадцатью однословными (по 32 бита) регистрами r0-r15, их назначение приведено в таблице 1.

Таблица 1: Распределение регистров процессора MIPT

r0-r12	свободно используются
r13	указатель фрейма вызова
r14	указатель стека
r15	счётчик команд
flags	результат операции сравнения

В зависимости от кода операции каждая команда может быть одного из следующих форматов:

- RM — 8 старших бит код команды, 4 следующих бита — код регистра (приёмника или источника), 20 младших бит — адрес в памяти в виде беззнакового числа от 0 до $2^{20} - 1$. Пример такой команды:

  ```
  load r0, 12323
  ```

- RR — 8 бит код команды, 4 бит код регистра-приёмника, 4 бит код регистра-источника, 16 бит модификатор источника, число со знаком от -2^{15} до $2^{15} - 1$. Примеры такой команды:

  ```
  mov r1, r2, -123
  ```

  ```
  add r1, r2, 0
  ```

  В командах такого типа непосредственный операнд всегда прибавляется к значению регистра-источника. Если этого делать не требуется, в поле непосредственного операнда заносится 0.

- RI — 8 бит код команды, 4 бит код регистра-приёмника, 20 бит непосредственный операнд, число со знаком от -2^{19} до 2^{19}. Пример такой команды:

  ```
  ori r2, 64
  ```

- J — 8 бит код команды, 4 бита игнорируются, 20 младших бит — адрес в памяти в виде беззнакового числа от 0 до $2^{20} - 1$. Пример такой команды:

  ```
  calli 3121
  ```

2 Описание процессора MIPT

2.1 Система команд процессора MIPT

Таблица 2: Описание машинных команд процессора M

Код	Имя	Формат	Описание
0	halt	RI	Останов процессора. `halt r1 0`
1	syscall	RI	Вызов операционной системы. `syscall r0, 100`
2	add	RR	Сложение регистров. К R_1 (регистру-первому аргументу) прибавляется содержимое R_2 (регистра второго аргумента), модифицированное непосредственным операндом. `add r1, r2, 3` К регистру `r1` прибавляется `r2+3`.
3	addi	RI	Сложение регистра с непосредственным операндом. К содержимому R_1 прибавляется значение Imm (непосредственного операнда). `addi r4, 10`
4	sub	RR	Вычитание регистров. Из R_1 вычитается содержимое R_2. `sub r3, r5, 5` Из регистра `r3` вычитается `r5+5`.
5	subi	RI	Вычитание из регистра непосредственного операнда. `subi r4 1`
6	mul	RR	Умножение регистров. Содержимого R_1 умножается на R_2. Результат помещается в пару регистров, начинающуюся с R_1. `mul r3, r10, 0` Результат произведения `r3` и `r10` будет помещён в `r3` и `r4`.

Продолжение на следующей странице

7	`muli`	`RI`	Умножение регистра R_1 на Imm. Результат помещается в пару регистров, начинающуюся с R_1. `muli r5, 100` После исполнения: в регистре `r5` младшие 32 разряда результата, в регистре `r6` — старшие 32 разряда.
8	`div`	`RR`	Деление регистров. Младшие 32 бита первого операнда находятся в регистре R_1, старшие — в регистре, номер которого на 1 больше R_1. Второй операнд находится в регистре. Частное помещается в регистр R_1, остаток — в следующий регистр. `div r3, r10, 0` `r3` будет содержать частное от деления пары `(r3,r4)` на `r10`, `r4` — остаток от этого деления.
9	`divi`	`RI`	Деление регистра на непосредственный операнд. Аналогично `div`. `div r3 10` `r3` будет содержать частное от деления пары `(r3,r4)` на `10`, `r4` — остаток от этого деления.
12	`lc`	`RI`	Загрузка константы Imm в регистр R_1. Для загрузки констант, бóльших 2^{20} требуются дополнительные команды сдвига и логического сложения. `lc r7, 123` `r7` будет содержать `123`.
13	`shl`	`RR`	Сдвинуть биты в регистре R_1 влево на значение регистра R_2. `shl r1, r2, 0`
14	`shli`	`RI`	Сдвинуть биты в регистре R_1 влево на Imm. `shli r1, 2`
15	`shr`	`RR`	Сдвинуть биты в регистре R_1 вправо на значение регистра R_2. Сдвижка на 32 или более разрядов обнуляет регистр R_1. `shr r1, r2, 0`
16	`shri`	`RI`	Сдвинуть биты в регистре R_1 впрево на Imm. `shri r1, 2`

Продолжение на следующей странице

17	`and`	`RR`	Логическое И регистров R_1 и R_2. Результат — в регистре R_1. `and r1 r2 0`
18	`andi`	`RI`	Логическое И над регистром R_1 и Imm. `andi r1, 255`
19	`or`	`RR`	Логическое ИЛИ регистров R_1 и R_2. Результат — в регистре R_1. `or r1, r2, 0`
20	`ori`	`RI`	Логическое ИЛИ над регистром R_1 и Imm. `ori r1, 255`
21	`xor`	`RR`	Логическое исключающее ИЛИ регистров R_1 и R_2. Результат — в регистре R_1. `xor r1 r2 0`
22	`xori`	`RI`	Логическое исключающее ИЛИ над регистром R_1 и Imm. `xori r1 255`
23	`not`	`RI`	Поразрядное НЕ над всеми битами R_1. Непосредственный операнд игнорируется, но он присутствует в команде для простоты компилятора. `not r1, 0`
24	`mov`	`RR`	Пересылка из регистра R_2 в регистр R_1. `mov r0, r3, 22` В регистр `r0` помещается значение `r3+22`.
32	`addd`	`RR`	Вещественное сложение регистров R_0 и R_1. Вещественные числа занимают пару регистров, младшие из которых фигурируют в коде операции. `addd r2, r5, 0` К вещественному числу, занимающему пару регистров `(r2,r3)` прибавляется значение вещественного числа из пары регистров `(r5,r6)`.
33	`subd`	`RR`	Вещественное вычитание регистров R_0 и R_1. `subd r2, r5, 0` Из вещественного числа, занимающему пару регистров `(r2,r3)` вычитается значение вещественного числа из пары регистров `(r5,r6)`.

Продолжение на следующей странице

34	`muld`	`RR`	Вещественное умножение регистров R_0 и R_1. `muld r2, r5, 0` Вещественное число, занимающее пару регистров (`r2,r3`) умножается на значение вещественного числа из пары регистров (`r5,r6`).
35	`divd`	`RR`	Вещественное деление регистров R_0 и R_1. `subd r2, r5, 0` Вещественное число, занимающее пару регистров (`r2,r3`) делится на значение вещественного числа из пары регистров (`r5,r6`).
36	`itod`	`RR`	Преобразование целого числа R_2 в вещественное R_1 `itod r2, r5, 0` Значение, находящееся в регистре `r5` преобразуется в вещественное и помещается в пару регистров (`r2,r3`).
37	`dtoi`	`RR`	Преобразование вещественного числа R_2 в целое R_1 `itod r2, r5, 0` Вещественное число, занимающее пару регистров (`r5,r6`) преобразуется в целое отбрасыванием дробной части. Если число не помещается в регистр, возникает ошибка.
38	`push`	`RI`	Отправить значение, находящееся в регистре R_1 с добавлением к нему Imm в стек. `push r0, 255` Поместить в стек число, равное `r0+255`. Указатель стека (`r14`) уменьшится на `1`.

39	`pop`	`RI`	Извлечь из стека находящееся там число, поместить его в регистр R_1 и затем прибавить к регистру R_1 значение Imm.
			`push r2, 0`
			`pop r3, 1`
			После исполнения в `r3` будет содержаться `r2+1`.

Продолжение на следующей странице

40	`call`	`RR`	Вызвать функцию, адрес которой расположен в $R_2 + Imm$. `call r0, r5, 0` Вызвать функцию, адрес которой извлекается из регистра `r5`. При вызове функции в стек помещается адрес команды, следующей за текущей и управление передаётся по указанному адресу. Этот же адрес помещается и в регистр `r0`.
41	`calli`	`J`	Вызвать функцию, адрес которой расположен в $imm20$. `call 13323` Вызвать функцию, начинающуюся с оператора по адресу `13323`. При вызове функции в стек помещается адрес команды, следующей за текущей и управление передаётся по указанному адресу.
42	`ret`	`RI`	Возврат из функции. `ret r0, 0` Вернуться из вызванной функции в вызвавшую. Из стека извлекается адрес, по которому передаётся управление и который будет адресом следующего исполняемого слова. Непосредственный аргумент показывает количество дополнительных слов, на которое требуется продвинуть указатель стека (оно должно соответствовать количеству аргументов при вызове функции).
43	`cmp`	`RR`	Сравнение. `cmp r0, r1, 0` Сравнить `r0` с `r1`. Результат записывается в регистр флагов, который используется в командах перехода.
44	`cmpi`	`RI`	Сравнение с константой. `cmpi r0, 0` Сравнить `r0` с нулём. Установить соответствующий флаг.

Продолжение на следующей странице

45	`cmpd`	`RR`	Сравнение вещественных чисел. `cmpd r1, r4, 0` Сравнить пару `(r1,r2)` с `(r4,r5)`. Установить соответствующий флаг.
46	`jmp`	`J`	Безусловный переход. `jmp 2212` Следующая исполняемая команда будет находиться по адресу 2212.
47	`jne`	`J`	Переход при наличии условия `!=`. `jne 2212` Если регистр флагов содержит условие `!=`, то следующая исполняемая команда будет находиться по адресу 2212 иначе ход исполнения программы не нарушается.
48	`jeq`	`J`	Переход при наличии условия `==`. `jne 2212` Если регистр флагов содержит условие `==`, то следующая исполняемая команда будет находиться по адресу 2212 иначе ход исполнения программы не нарушается.
49	`jle`	`J`	Переход при наличии условия `<=`. `jle 2212` Если регистр флагов содержит условие `<=`, то следующая исполняемая команда будет находиться по адресу 2212 иначе ход исполнения программы не нарушается.
50	`jl`	`J`	Переход при наличии условия `<`. `jl 2212` Если регистр флагов содержит условие `<`, то следующая исполняемая команда будет находиться по адресу 2212 иначе ход исполнения программы не нарушается.

Продолжение на следующей странице

51	`jge`	`J`	Переход при наличии условия >=. `jne 2212` Если регистр флагов содержит условие >=, то следующая исполняемая команда будет находиться по адресу 2212 иначе ход исполнения программы не нарушается.
52	`jg`	`J`	Переход при наличии условия >. `jg 2212` Если регистр флагов содержит условие >, то следующая исполняемая команда будет находиться по адресу 2212 иначе ход исполнения программы не нарушается.
64	`load`	`RM`	Загрузка из памяти в регистр `load r0, 12345` Содержимое из ячейки памяти по адресу 12345 копируется в регистр `r0`.
65	`store`	`RM`	Выгрузка из регистра в память `store r0, 12344` Содержимое из регистра `r0` копируется в ячейку памяти по адресу 12344.
66	`load2`	`RM`	Загрузка из памяти в пару регистров `load r0, 12345` Содержимое из ячеек памяти по адресу 12345 и 12346 копируется в регистры `r0` и `r1`.
67	`store2`	`RM`	Выгрузка из пары регистров в память `load r0, 12345` Содержимое из регистра `r0` копируется по адресу 12345, а из ячейки `r1` — по адресу 12346.
68	`loadr`	`RR`	Загрузка из памяти в регистр `load r0, r1, 15` Содержимое из ячейки памяти по адресу `(r1+15)` копируется в регистр `r0`.
69	`storer`	`RR`	Выгрузка из регистра в память `store r0, r13, 3` Содержимое из регистра `r0` копируется в ячейку памяти по адресу `r13+3`.

Продолжение на следующей странице

70	`loadr2`	`RR`	Загрузка из памяти в пару регистров `load r0, r13, 7` Содержимое из ячеек памяти по адресу `r13+7` и `r13+8` копируется в регистры `r0` и `r1`.
71	`storer2`	`RR`	Выгрузка из пары регистров в память `load r0, r13, 11` Содержимое из регистра `r0` копируется по адресу `r13+11`, а из ячейки `r1` — по адресу `r13+12`.

4 Примеры программ с комментариями

4.1 Простая программа возведения числа в квадрат

```
main:
    syscall r0, 100  ;ввод числа со стандартного ввода в регистр r0
    mov r2, r0, 0     ;копирование регистра r0 в регистр r2
    mul r0, r2, 0     ;пара регистров (r0,r1) содержит произведение
    syscall r0, 102  ;вывод содержимого регистра r0 (младшая часть)
    lc r0, 10        ;загрузка константы 10 ('\n') в регистр r0
    syscall r0, 105  ;вывод '\n'
    lc r0, 0         ;
    syscall r0, 0    ;выход из программы с кодом 0
    end main        ;начать исполнение с main
```

4.2 Программа возведения числа в квадрат, использующая функции

```
sqr:                ;функция sqr с одним аргументом в стеке
    loadr r0, r14, 1  ;загрузка r0 ячейки с первым аргументом
    mov r2, r0, 0     ;копируем r0 в r2
    mul r0, r2, 0     ;(r0,r1) = r0*r2
    ret 1           ;возвращаемся из функции, убрав аргумент из стека
intout:             ;функция, распечатывающая аргумент + '\n'
    load r0, r14, 1   ;загрузка первого аргумента в r0
    syscall r0, 102  ;вывод r0 на экран
    lc r0, 10        ;загрузка '\n'
    syscall r0, 105  ;вывод '\n'
    ret 1           ;возврат
main:
    syscall r0, 100  ;считывание в r0
    push r0, 0       ;помещение r0+0 в стек
    calli sqr       ;вызов функции sqr. В r0 функция оставит результат.
    push r0, 0       ;передаём результат в  фунцкию intout
    calli intout    ;и вызываем её
    lc r0, 0
    syscall r0, 0    ;exit(0)
    end main
```

4.3 Программа, использующая рекурсивную функцию факториала

```
fact:
    loadr r0, r14, 1
    cmpi r0, 1
    jg skip0
    lc r0, 1
    ret 1
skip0:
    push r0, 0
    subi r0, 1
    push r0, 0
    calli fact
    pop r2, 0
    mul r0, r2, 0
    ret 1
main:
    syscall r0, 100
    push r0, 0
    calli fact
    syscall r0, 102
    lc r0, 10
    syscall r0, 105
    lc r0, 0
    syscall r0, 0
    end main
```

4.4 Программа, вводящая два вещественных числа и печатающая их сумму, разность, произведение и частное

```
fout:
    syscall r0, 103
    lc r0, 10
    syscall r0, 105
    ret 0
main:
    syscall r2, 101
    syscall r4, 101

    mov r0, r2, 0
    mov r1, r3, 0
    addd r0, r4, 0
    calli fout
    mov r0, r2, 0
    mov r1, r3, 0
    subd r0, r4, 0
    calli fout
    mov r0, r2, 0
    mov r1, r3, 0
    muld r0, r4, 0
    calli fout
    mov r0, r2, 0
    mov r1, r3, 0
    divd r0, r4, 0
    calli fout
    lc r0, 100
    itod r0, r0, 0
    calli fout
    lc r0, 0
    syscall r0, 0
end main
```