2-й семестр. Домашние задания.

1 Архитектура компьютера MIPT64

`MIPT64` — машина с архитектурой Фон-Неймана с адресным пространством в 2^{42} байт. Для моделирование ограничим адресное пространство в 2^{21} байт.

Каждая команда занимает ровно одно слово, 6 старших бит которого — код операции, а использование остальных 26 бит зависит от операции.

У компьютера 32 целочисленных регистра `r0-r31`, каждый по 64 бита, их назначение приведено в таблице 1. Эти регистры могут содержать и вещественные числа.

Таблица 1: Распределение регистров процессора `MIPT64`

`r0-r26`	свободно используются
`r27`	всегда содержит 0. Другое название — `rz`
`r28`	указатель фрейма вызова. Другое название — `fp`
`r29`	регистр стека. Другое название — `sp`
`r30`	регистр связи. Другое название — `lr`
`r31`	регистр команды. Другое название — `pc`
`flags`	результат последней операции сравнения

Хотя регистры 64-битные, каждая машинная команда содержит ровно по 32 бита.

Внимание: все адреса в программах выражаются не в словах, а в байтах!

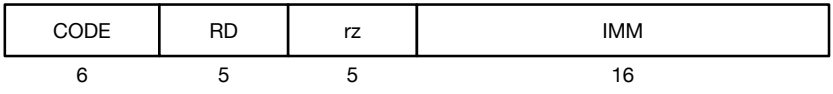
В зависимости от кода операции каждая команда может быть одного из следующих форматов:

- `RR`.

  Все команды этого формата — трёхоперандные. Всегда имеется регистр-приёмник и регистр-источник. Третий операнд зависит от регистра-источника

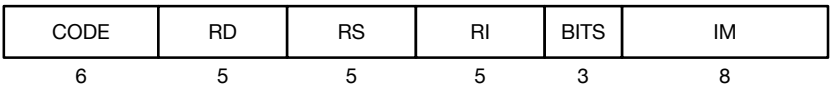
  6 старших бит — код команды, 5 бит — код регистра-приёмника, 5 бит —код регистра-источника.

  - Если регистр источника — `rz`, то следующие `16` бит — `IMM`, непосредственная часть операнда со знаком.



– Если регистр источника `SRC` отличен от `rz`, то далее следует 5 бит — номер индексного регистра, далее — 3 бита `BITS`, на которые сдвигается значение индексного регистра, а последние 8 бит — непосредственный операнд `IM` со знаком.

Тогда второй операнд равен `(RI≪BITS)+IM`.



Примеры команд:

`add r1,r2,r3,2,0` — присвоить регистру `r1` значение, равное `r2+(r3<<2)`.

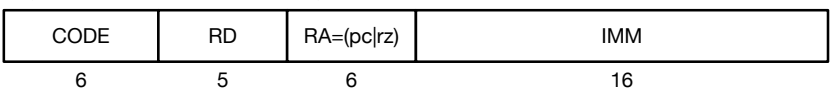
`add r0,rz,rz,0,0` — присвоить регистру `r0` значение 0.

`add r0,pc,rz,0,-12` – присвоить регистру `r0` значение, равное текущему счётчику команд минус 12.

- `RM`. 6 старших бит — код команды. 5 следующих бит — код регистра (приёмника или источника). Затем следует 5 бит номера регистра, от которого вычисляется адрес (будем называть его *адресный регистр* или, сокращённо, `RA`.

  Имеются три варианта команды:

  – Если `RA == pc или RA == rz`, то последние 16 бит — число со знаком, добавляемое к значению регистра. В результате получается *исполнительный адрес*, по которому или кладётся значение регистра-источника (приёмником тогда является память), либо откуда извлекается значение и кладётся в регистр приёмника. Если регистр равен `rz`, то 16 бит беззнаковые, иначе знаковые.

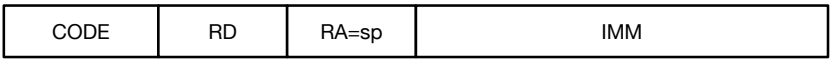


  Пример команды:

  `ld r5, pc, -32` — загрузить в регистр `r5` значение, находящееся по адресу `pc-32`.

  `ld r5, rz, 23132` — загрузить в регистр `r5` значение, находящееся по абсолютному адресу `23132`.

  – Если `RA == sp`, то следующие 16 бит — количество байт `IMM` (без знака), на которые изменится регистр `sp` после исполнения команды. При загрузке в стек указатель стека уменьшается (`push`) *до* совершения операции (предекремент), при выгрузке из стека — увеличивается (`pop`) *после* совершения операции.



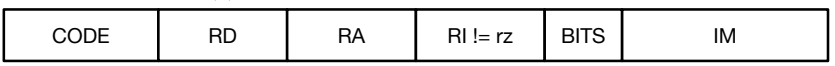
Стек должен всегда быть выровнен на границу 4 байт.

Пример команды:

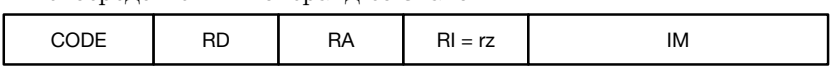
`st r3, sp, 8` — уменьшить значение стека на 8, положить в стек значение регистра `r3`.

`ld r3, sp, 8` — положить в регистр `r3` значение, находящееся на стеке, увеличить значение стека на 8.

– Иначе следующие 5 бит — номер индексного регистра (`RI`), следующие 3 бита — число бит `BITS`, на которые сдвигается значение индексного регистра, а последние 8 бит — непосредственный операнд `IM` со знаком. Тогда `EA = RA+(RI<<BITS)+IM`



Если же индексный регистр равен регистру `rz`, то последние 11 бит — непосредственный операнд со знаком.



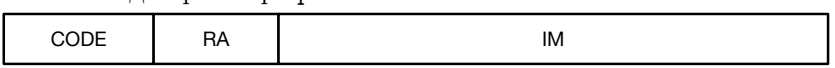
Примеры такой команды:

`ld r5,fp,rz,-12` — загрузить в регистр `r5` значение по адресу `fp-12`.

`ld r5,r11,r4, 3, 0` — загрузить в регистр `r5` значение по адресу `[r11+r4*8]`. Эта команда загружает в регистр значение элемента массива, указатель на который находится в регистре `r11` с индексом `r4`.

- `B`. Код команды — 6 старших бит, 5 бит — код адресного регистра `RA`.

  – Если адресный регистр равен `pc` или `rz`, то следующие 21 бит — непосредственное число без знака, если адресный регистр `rz` и число со знаком для регистра `pc`.

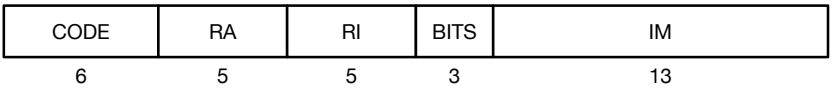


`bl rz, out1` — вызов функции под именем `out1`. Используется абсолютная адресация. При такой адресации невозможно вызвать функцию, имеющую адрес более $2^{21} - 1$.

`bl pc, out1` — вызов функции под именем `out1`. Используется адресация относительно счётчика команд.

`bl out1` — вызов функции под именем `out1`. Используется адресация относительно счётчика команд.

– Если адресный регистр не равен ни `pc`, ни `rz`, то адрес вызова равен `RA+(RI<<BITS)+IMM`.



Программа вводит число с stdin, прибавляет 1 и выводит на stdout.

```
main:
        svc r0, rz, 100
        add r0, r0, rz, 0, 1 ; r0++
        svc r0, rz, 102
        svc r0, rz, 0
        end main
```

2 Описание процессора MIPT64

2.1 Система команд процессора MIPT64

Таблица 2: Описание машинных команд процессора MIPT64

Код	Имя	Формат	Описание
0	halt	RR	Останов процессора. halt r0, rz, 0
1	svc	RR	Вызов операционной системы. Значение адресного выражения — код вызова. svc r0, rz, 102
2	add	RR	Сложение. Регистр-приёмник принимает значение суммы регистра-источника и модифицирующей части. add r1, r2, r3, 0, 3 Регистр r1 принимает значение r2+r3+3. add r1, r1, rz, 0, 1 Регистр r1 принимает значение r1+1. add r0, rz, 0 Регистр r0 обнуляется.
3	sub	RR	Вычитание. Регистр-приёмник принимает значение разности регистра-источника и модифицирующей части. sub r1, r2, r3, 0, 3 Регистр r1 принимает значение r2-(r3+3). sub r5, r4, r3, 2, 1 Регистр r5=r4-(4*r3+1)
4	mul	RR	Умножение. Регистр-приёмник принимает значение младших 64 битов знакового произведения регистра-источника и модифицирующей части. mul r1, r2, rz, 0, 3 r1 = r2 * 3.

Продолжение на следующей странице

5	`div`	`RR`	Деление. Регистр-приёмник принимает значение беззнакового частного регистра-источника и модифицирующей части.
			`div r1, r1, r2, 5, -7`
			`r1 = r1 / (r2*32-7).`

6	`mod`	`RR`	Остаток от деления. Регистр-приёмник принимает значение остатка от беззнакового частного регистра-источника и модифицирующей части.
			`mod r1, r1, r3, 3, 4`
			`r1 = r1 % (r3*8+4).`

7	`and`	`RR`	Побитовое И. Регистр-приёмник принимает значение побитового `and` регистра-источника и модифицирующей части.
			`and r1, r1, rz, 0, 15`
			`r1 = r1 & 15.`

8	`or`	`RR`	Побитовое ИЛИ. Регистр-приёмник принимает значение побитового `or` регистра-источника и модифицирующей части.
			`or r1, r1, rz, 0, 64`
			`r1 = r1 | 64.`

9	`xor`	`RR`	Побитовое XOR. Регистр-приёмник принимает значение побитового `xor` регистра-источника и модифицирующей части.
			`xor r1, r1, rz, 0, 0`
			`r1 = 0.`

10	`nand`	`RR`	Побитовое NAND. Регистр-приёмник принимает значение побитового `not and` регистра-источника и модифицирующей части.
			`nand r1, r1, r2, 1, 6`
			`r1 = r1 &  (r2*2 + 6).`

11	`shl`	`RR`	Побитовый левый сдвиг. Регистр-приёмник принимает значение регистра-источника, сдвинутого влево на младшие 6 бит модифицирующей части.
			`shl r1, r1, rz, 0, 1`
			`r1 = r1 << 1.`

Продолжение на следующей странице

12	`shr`	`RR`	Побитовый правый сдвиг. Регистр-приёмник принимает значение регистра-источника, сдвинутого влево на младшие 6 бит модифицирующей части. `shr r1, r1, rz, 0, 1` `r1 = r1 >> 1`.
13	`addd`	`RR`	Вещественное сложение. Регистр-приёмник принимает значение суммы регистра-источника и модифицирующей части. `addd r1, r1, r2, 3, -7` `r1 = r1 + (r2 * 8 - 7)`.
14	`subd`	`RR`	Вещественное вычитание. Регистр-приёмник принимает значение разности регистра-источника и модифицирующей части. `subd r1, r2, r3, 0, 0` `r1 = r2 - r3`.
15	`muld`	`RR`	Вещественное умножение. Регистр-приёмник принимает значение произведения регистра-источника и модифицирующей части. `muld r1, r2, r3, 0, 0` `r1 = r2 * r3`.
16	`divd`	`RR`	Вещественное деление. Регистр-приёмник принимает значение частного регистра-источника и модифицирующей части. `divd r1, r2, r3, 0, 0` `r1 = r2 * r3`.

17	`itod`	`RR`	Преобразование из целого в вещественное. Регистр-приёмник принимает значение суммы регистра-источника и модифицирующей части.
			`itod r1, rz, 0, 0`
			`r1 = 0.`
			`itod r1, r2, rz, 0, 0`
			`r1 = (double)r2`.

18	`dtoi`	`RR`	Преобразование из вещественного в целое. Регистр-приёмник принимает значение суммы регистра-источника и модифицирующей части. `dtoi r1, r2, rz, 0, 0` `r1 = (int64)r2`.

Продолжение на следующей странице

19	`bl`	`B`	Вызов функции. Регистр `lr` принимает значение `pc`, после чего регистр `pc` принимает значение модифицирующей части.
			`bl rz, out1`
			Вызов функции `out1` в режиме абсолютной адресации.
			`bl pc, out1`
			Вызов функции `out1` в режиме адресации относительно регистра `pc`.
			`bl fact`
			Вызов функции `fact`. Компилятор сам выбирает подходящий режим адресации, `pc` или `rz`.
			`bl r5, rz, 0, 0`
			Вызов функции, адрес которой находится в `r5`.
			`bl r5, r3, 3, 0`
			Вызов функции, адрес которой находится в `r5+r3*8`.

20	`cmp`	`RR`	Сравнение целочисленное. Регистр `flags` принимает значение результата сравнения. Формат команды совпадает с форматом команды `sub` за исключением того, что результат никуда не записывается.
			`cmp r0, rz, 0`
			Сравнить `r0` с нулём.
			`cmp r1, r2, rz, 0, 0`
			Сравнить `r1` и `r2`

21	`cmpd`	`RR`	Сравнение вещественное. Регистр `flags` принимает значение результата сравнения. Формат команды совпадает с форматом команды `sub` за исключением того, что результат никуда не записывается. `cmpd r1, r2, rz, 0, 0` Сравнить `r1` и `r2`

Продолжение на следующей странице

22	`cne`	`RR`	Условное присваивание регистров по `flags`. Если условие `!=` удовлетворяется, то происходит присвоение регистру приёмнику результата сложения регистра-источника и модифицирующей части.
			`cne r1, r2, rz, 0, 0`
			При неравенстве присвоить регистру `r1` значение `r2`
			`cne pc, fast_way`
			При неравенстве присвоить регистру `pc` значение метки `fast_way`, то есть условно перейти по адресу `fast_way`.

23	`ceq`	`RR`	Условное присваивание регистров по `flags`. Если условие `==` удовлетворяется, то происходит присвоение регистру приёмнику результата сложения регистра-источника и модифицирующей части.
			`ceq r1, r2, rz, 0, 0`
			При равенстве присвоить регистру `r1` значение `r2`
			`cne pc, lab1`
			При неравенстве присвоить регистру `pc` значение `lab1`, то есть условно перейти по адресу `lab1`

24	`cle`	`RR`	Условное присваивание регистров по `flags`. Если условие `<=` удовлетворяется, то происходит присвоение регистру приёмнику результата сложения регистра-источника и модифицирующей части. `cle r1, r2, rz, 0, 0` При условии `<=` присвоить регистру `r1` значение `r2`
25	`clt`	`RR`	Условное присваивание регистров по `flags`. Если условие `<` удовлетворяется, то происходит присвоение регистру приёмнику результата сложения регистра-источника и модифицирующей части. `clt r1, r2, rz, 0, 0` При условии `<` присвоить регистру `r1` значение `r2`
26	`cge`	`RR`	Условное присваивание регистров по `flags`. Если условие `>=` удовлетворяется, то происходит присвоение регистру приёмнику результата сложения регистра-источника и модифицирующей части. `cge r1, r2, rz, 0, 0` При условии `>` присвоить регистру `r1` значение `r2`

Продолжение на следующей странице

27	`cgt`	`RR`	Условное присваивание регистров по `flags`. Если условие `>` удовлетворяется, то происходит присвоение регистру приёмнику результата сложения регистра-источника и модифицирующей части. `cgt r1, r2, rz, 0, 0` При условии `>` присвоить регистру `r1` значение `r2`
28	`ld`	`rm`	Загрузка регистра из памяти `ld r1, sp, 8` Загрузить из стека `r1`, затем увеличить стек на 8. `ld r5, pc, -32` Загрузить в `r5` значение по адресу `pc-32`
29	`st`	`rm`	Выгрузка регистра в память `st r5, sp, 8` Уменьшить значение стека на 8, затем положить туда `r5`.

Таблица 3: Коды системных вызовов операционной системы FUMPOS

Код	Аргументы	Описание
0	EXIT	код возврата.
100	SCANINT	
101	SCANDOUBLE	
102	PRINTINT	
103	PRINTDOUBLE	
104	GETCHAR	
105	PUTCHAR	C

Для удобства все коды команд собраны в перечислимый тип code:

```
enum code_e {
     HALT = 0,
     SVC = 1,
     ADD = 2,
     SUB = 3,
     MUL = 4,
     DIV = 5,
     MOD = 6,
     AND = 7,
     OR = 8,
     XOR = 9,
     NAND = 10,
     ADDD = 12,
     SUBD = 13,
     MULD = 14,
     DIVD = 15,
     ITOD = 16,
     DTOI = 17,
     BL = 20,
     CMP = 22,
     CMPD = 23,
     CNE = 25,
     CEQ = 26,
     CLE = 27,
     CLT = 28,
     CGE = 29,
     CGT = 30,
     LD = 31,
     ST = 32,
 };
```

2.2 Особенности некоторых команд

2.2.1 Общие особенности

Процессор содержит минимальный набор команд и не имеет явных команд копирования регистров, условных и безусловных переходов, возврата из функции. Команды имеют сложный формат, поля которых зависят от других полей. Единым полем является 6-битный код операции.

Вещественные и целочисленные регистры разделяют одни и те же регистры. В одних командах биты этих регистров трактуются как образ целого 64-битного числа со знаком, в других — как образы 64-битных вещественных чисел.

Для явной работы с памятью имеется ровно две команды — `ld` и `st`. Эти команды имеют формат `RM`. Почти все остальные команды имеют формат `RR`.

Имеется специальный регистр, который всегда содержит нуль, `rz` или `r27`, он участвует в создании непосредственных операндов.

При исполнении каждой инструкции можно явно использовать регистр `pc`. Он всегда равен адресу *следующей* за текущей инструкции. Это оказывается важным, например, при инструкциях перехода и вызова функций.

2.2.2 О формировании модифицирующей части команды

Можно сказать, что все команды формата `RR` имеют три операнда: регистр-приёмник `RD`, регистр-источник `RS` и модифицирующая часть, которая занимает 16 бит и может иметь различный формат. Если регистр-источник — регистр `rz`, то модифицирующая часть — 16-битное число *со знаком*, то есть `111111111111` в этих 16 битах должно восприниматься как `-1`.

Программа может содержать метки — адреса, по которым могут вызываться функции, происходить переходы и находиться данные.

Для удобства программирования на ассемблере, в командах условного присваивания `CNE,CGT...` правая часть может состоять только из метки. Ассемблер должен самостоятельно выбрать такой способ адресации, при котором значение адресуемого выражения было бы равно значению этой метки (например, относительно `pc`).

2.3 Вызовы функций и возвраты из них

Функция вызывается инструкцией `bl`. В регистр `lr` отправляется содержимое счётчика команд `pc`, а в регистр `pc` — адресное выражение инструкции. Следующая инструкция должна выбираться процессором по адресу `pc`, а так как `pc` меняется, происходит переход по новому адресу.

Если исполняемая функция не вызывает другие (*листовая*), то регистр `lr` сохранять не требуется и возврат из функции —просто замена регистра `pc` регистром `lr`.

```
add pc, lr, rz, 0, 0  ;  возврат из функции
```

Если вызывает — регистр `lr` необходимо сохранить, возможно, в стеке.

При сокращённом варианте вызова функции регистр `pc` может опускаться.

```
bl fibo
```

2.3.1 Работа со стеком

Инструкции `ld` и `sp` умеют работать с любыми регистрами, содержащими адрес. Но если адресный регистр есть регистр `sp`, то при его использовании он автоинкрементируется или автодекрементируется на указанное в непосредственном операнде число байт.

Если принять стек *sp как указатель на long long (на самом деле `sp` адресует байты), то:

```
st  lr, sp, 8   ; *--sp = lr
ld  lr, sp, 8   ; lr = *sp++;
```

2.3.2 Вещественная арифметика

Регистры могут содержать как целые, так и вещественные значения. Трактовка содержащихся в них бит зависит только от операции.

При работе с вещественными числами регистр `rz` также содержит нулевое значение. Операция сдвига регистра влево в модифицирующей части — умножение его на соответствующую степень двойки. Операция сложения тоже производится с вещественным значением.

2.3.3 Дополнительные директивы

Имеется 4 дополнительных директивы, резервирующие слова памяти. Перед ними можно поставить метки, чтобы их адресовать.

- `word 123` — зарезервировать 32 бита и заполнить их значением `123`.
- `dword 321` — зарезервировать 64 бита и заполнить их значением `321`.
- `double 3.1415926` — зарезервировать 64 бита и заполнить их значением `3.1415926`.
- `bytes 123` — зарезервировать 123 байта и заполнить их значением `0`. Выровнять следующие данные на границу, кратную 4. В данном случае зарезервируется 124 байта.

2.3.4 Примеры программ

```
; Рекурсивный факториал
in0: ; ввести integer с stdin в r0
        svc r0,rz,100
        add pc, lr, rz, 0, 0 ; вернуться по lr

out0: ; вывести регистр r0 на stdout
        svc r0,rz,102
        add pc, lr, rz, 0, 0

exit: ; завершить программу.
        svc r0,rz,0

fact:   ; сам факториал. В r0 - аргумент n
        cmp r0,rz,1     ; сравниваем r0 с 1
        cgt pc, rec     ; > ? тогда pc=rec
        add pc, lr, rz, 0, 0 ; return
rec:
        st  lr, sp, 8   ; lr - в стек
        st  r0, sp, 8   ; sp - тоже
        sub r0, r0, rz, 0, 1 ; r0--
        bl  fact        ; fact(n-1)
        ld  r1, sp, 8   ; верхушка стека -> r1
        ld  lr, sp, 8   ; восстанавливаем lr
        mul r0, r0, r1, 0, 0 ; r0 *= r1
        add pc, lr, rz, 0, 0 ; return

main:   ; вызывалка всего
        bl in0          ; r0 = in0()
        bl fact         ; r0 = fact(r0)
        bl out0         ; out0(r0)
        bl exit         ; exit(r0)
        end main
```

```
; вычисление n-го Фибоначчи с помощью динамического программирования.
fibo:
    cmp r0,rz,2
    cge pc,norec
    add pc,lr,rz,0,0
norec:
    add r2,pc,cache ; r2 = \&cache
    ; check if already solved
    ld r1, r2, r0, 3, 0 ; r1=cache[n]
    cmp r1,rz,0
    cgt pc,fast_way
slow_way:
    st lr, sp, 8    ; ret
    st r0, sp, 8    ; save n
    st r0, sp, 8    ; twice
    sub r0,r0,rz,0,1 ; n--
    bl fibo ;  r0=fibo(n-1)
    ld r1, sp, 8 ; r1=n
    st r0, sp, 8 ; push fibo(n-1)
    sub r0, r1, rz, 0, 2 ; r0=n-2
    bl fibo         ; r0=fibo(n-2)
    ld r1, sp, 8 ; r1 = fibo(n-1)
    add r0, r0, r1, 0, 0 ; r0=fibo(n-1)+fibo(n-2)
    ld r1, sp, 8   ; n
    st r0, r2, r1, 3, 0 ; cache[n] = ret
    ld pc, sp, 8    ; return to lr = top
fast_way:
    add r0,r1,rz,0,0
    add pc,lr,rz,0,0

main:
    svc r0, rz, 100
    bl fibo
    svc r0, rz, 102
    add r0,rz,10
    svc r0,rz,105
    svc r0, rz, 0

cache:
    bytes 800   ; unsigned long long [100]

    end main
```