

Федеральное государственное автономное образовательное учреждение высшего
образования

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ДОМАШНЯЯ РАБОТА ПО ДИСЦИПЛИНЕ

Схемотехника ЭВМ
Решение упражнений

Проверил:

.....

«_____» _____ 2021г.

Оценка _____

Выполнил:

Студент группы Р3455

Федюкович С. А. _____

6 Архитектура

Упражнение 6.1

Условие

Приведите три примера из архитектуры MIPS для каждого из принципов хорошей разработки:

1. Для простоты придерживайтесь единообразия;
2. Типичный сценарий должен быть быстрым;
3. Чем меньше, тем быстрее;
4. Хорошая разработка требует хороших компромиссов.

Поясните, как каждый из ваших примеров иллюстрирует соответствующий принцип.

Ответ

1. Для простоты придерживайтесь единообразия:

MIPS имеет только три типа инструкций(R, I и J).

Каждая инструкция имеет одинаковый номер и порядок операндов. А так же и одинаковый размер(6 бит), что делает процесс декодирования простым.

2. Типичный сценарий должен быть быстрым:

Благодаря регистрам доступ к часто используемым переменным осуществляется быстро. А благодаря RISC архитектуре ЭВМ работает только с небольшими и простыми инструкциями, что улучшает производительность.

Все инструкции в 32 битах. Данный размер специально выбран для ускорения типичных сценариев.

3. Чем меньше, тем быстрее:

В регистровом файле всего 32 регистра.

SPU ISA включает только небольшое количество инструкций, что позволяет сделать аппаратное обеспечение маленьким и соответственно быстрым. Сами же инструкции занимают небольшое количество памяти, что дает возможность загружать их быстрее.

4. Хорошая разработка требует хороших компромиссов:

MIPS использует три типа инструкций вместо одного.

По хорошему, все доступы должны выполняться так же быстро, как и доступ к регистру, но MIPS архитектура так же поддерживает доступ к ОЗУ, чтобы дать выбор между быстрым доступом и большим количеством памяти.

Та как MIPS это RISC архитектура, то она включает только набор простых инструкций, она же предоставляет псевдокод и его компилятор для часто используемых операций(перемещение данных из одного в регистра в другой и загрузка 32 бит немедленно).

Упражнение 6.2

Условие

Архитектура MIPS содержит набор 32-битных регистров. Можно ли создать компьютерную архитектуру без регистров? Если можно, кратко опишите такую архитектуру и её систему команд. Каковы преимущества и недостатки будут у этой архитектуры по сравнению с архитектурой MIPS?

Ответ

Да, можно спроектировать компьютерную архитектуру без набора регистров. Например, архитектура может использовать память как очень большой набор регистров. Каждая инструкция потребует доступа к памяти. Например, инструкция сложения может выглядеть так: `add 0x10, 0x20, 0x24`.

Такая инструкция сложила бы значения, хранящиеся по адресам памяти `0x20` и `0x24`, и поместила бы результат в ячейку с адресом `0x10`. Другие инструкции будут иметь тот же шаблон, доступ к памяти вместо регистров. Некоторые преимущества архитектуры заключаются в том, что для неё требуется меньше инструкций. Загрузки и хранения операций теперь не нужны. Это упростит и ускорит аппаратное декодирование.

Некоторые недостатки этой архитектуры по сравнению с архитектурой MIPS заключаются в том, что каждая операция потребует доступа к памяти. Таким образом, либо процессор должен быть медленным, либо он должен иметь малый объем памяти. Кроме того, поскольку инструкции должны кодировать адреса памяти вместо номеров регистров, размер инструкций будет большим для доступа ко всем адресам памяти. Или, альтернативно, каждая инструкция может получить доступ только к меньшему количеству адресов памяти. Например, архитектура может требовать, чтобы один из исходных операндов также был операндом-адресатом, уменьшая количество адресов памяти, которые должны быть закодированными.

Упражнение 6.3

Условие

Представьте себе 32-битное слово, хранящееся в адресуемой побайтово памяти и имеющее порядковый номер 42.

1. Каков байтовый адрес у слова памяти с порядковым номером 42?
2. Каковы все байтовые адреса, занимаемые этим словом памяти?
3. Предположим, что в этом слове хранится значение `0xFF223344`. Изобразите графически, как это число хранится в байтах слова в случаях прямого и обратного порядка следования байтов. Отметьте байтовые адреса всех байтов данных.

Ответ

1. $42 \times 4 = 168 = 10101000_2 = 0xA8$.
2. `0xA8` через `0xAB`.
- 3.

	Big-Endian					Little-Endian				
Byte Address	A8	A9	AA	AB	Word Address	AB	AA	A9	A8	Byte Address
Data Value	FF	22	33	44	0x48	FF	22	33	44	Data Value
	MSB		LSB			MSB		LSB		

Упражнение 6.4

Условие

Повторите Упражнение 6.3 для 32-битного слова, имеющего порядковый номер 15 в памяти.

Ответ

1. $15 \times 4 = 60 = 1111100_2 = 0x3C$.
2. `0x3C` через `0x3F`.
- 3.

	Big-Endian					Little-Endian				
Byte Address	3C	3D	3E	3F	Word Address	3F	3E	3D	3C	Byte Address
Data Value	FF	22	33	44	0x48	FF	22	33	44	Data Value
	MSB		LSB			MSB		LSB		

Упражнение 6.5

Условие

Объясните, как использовать следующую программу, чтобы определить, является ли порядок следования байтов прямым или обратным:

```
li $t0, 0xABCD9876
sw $t0, 100($0)
lb $s5, 101($0)
```

Ответ

```
# Big-endian
li $t0, 0xABCD9876
sw $t0, 100($0)
lb $s5, 101($0) # LSB от $s5 = 0xCD
# Little-endian
li $t0, 0xABCD9876
sw $t0, 100($0)
lb $s5, 101($0) # LSB от $s5 = 0x98
```

В формате с прямым порядком байтов байты пронумерованы от 100 до 103 слева направо. верно. В обратом байты нумеруются от 100 до 103 справа налево. Таким образом, команда последнего байта загрузки (lb) возвращает другое значение в зависимости от порядка байтов машины.

Упражнение 6.6

Условие

Используя кодировку ASCII, запишите следующие строки в виде последовательностей шестнадцатеричных значений символов этих строк (прим. переводчика: вам, вероятно, потребуется транслитерировать ваше имя, используя латинский алфавит).

1. SOS
2. Cool!
3. Semyon

Ответ

1. 0x53 4F 53 00
2. 0x43 6F 6F 6C 21 00
3. 0x53 65 6D 79 6F 6E 00

Упражнение 6.7

Условие

Повторите Упражнение 6.6 для следующих строк:

1. howdy
2. ions
3. To the rescue!

Ответ

1. 0x68 6F 77 64 79 00
2. 0x6C 69 6F 6E 73 00
3. 0x54 6F 20 74 68 65 20 72 65 73 63 75 65 21 00

Упражнение 6.8

Условие

Покажите, как строки из Упражнение 6.6 хранятся в адресуемой побайтово памяти, начиная с адреса 0x1000100C:

1. на машине с прямым порядком следования байтов;
2. на машине с обратным порядком следования байтов.

Отметьте байтовые адреса всех байтов данных в обоих случаях.

Ответ

1. Big-Endian

Word Address	Data	Word Address	Data	Word Address	Data
.
1000100C	53 4F 53 00	10001010	21 00	10001010	6F 6E 00
.	Byte 0 . Byte 3	1000100C	43 6F 6F 6C	1000100C	53 65 6D 79
.		.	Byte 0 . Byte 3	.	Byte 0 . Byte 3
(1)		(2)		(3)	

2. Little-Endian

Word Address	Data	Word Address	Data	Word Address	Data
.
1000100C	00 53 4F 53	10001010	00 21	10001010	00 6E 6F
.	Byte 3 . Byte 0	1000100C	6C 6F 6F 43	1000100C	79 6D 65 53
.		.	Byte 3 . Byte 0	.	Byte 3 . Byte 0
(1)		(2)		(3)	

Упражнение 6.9

Условие

Повторите Упражнение 6.8 для строк из Упражнение 6.7.

Ответ

1. Big-Endian

Word Address	Data	Word Address	Data	Word Address	Data
.
.	.	.	.	10001018	65 21 00
.	.	.	.	10001014	65 73 63 75
10001010	79 00	10001010	73 00	10001010	68 65 20 72
1000100C	68 6F 77 64	1000100C	6C 69 6F 6E	1000100C	54 6F 20 74
.	Byte 0 . Byte 3	.	Byte 0 . Byte 3	.	Byte 0 . Byte 3
(1)		(2)		(3)	

2. Little-Endian

Word Address	Data	Word Address	Data	Word Address	Data
.
.	.	.	.	10001018	00 21 65
.	.	.	.	10001014	75 63 73 65
10001010	00 79	10001010	00 73	10001010	72 20 65 68
1000100C	64 77 6F 68	1000100C	6E 6F 69 6C	1000100C	74 20 6F 54
.	Byte 3 . Byte 0	.	Byte 3 . Byte 0	.	Byte 3 . Byte 0
(1)		(2)		(3)	

Упражнение 6.10

Условие

Преобразуйте следующий код из языка ассемблера MIPS в машинный язык. Запишите инструкции в шестнадцатеричном формате.

```
add $t0, $s0, $s1
lw  $t0, 0x20($t7)
addi $s0, $0, -10
```

Ответ

```
0x02114020
0x8de80020
0x2010fff6
```

Упражнение 6.11

Условие

Повторите Упражнение 6.10 для следующего кода:

```
addi $s0, $0, 73
sw   $t1, -7($t2)
sub  $t1, $s7, $s2
```

Ответ

```
0x20100049
0xad49fff9
0x02f24822
```

Упражнение 6.12

Условие

1. Какие инструкции из Упражнение 6.10 являются инструкциями типа *I*?
2. Для каждой инструкции типа *I* из Упражнение 6.10 примените расширение знака к непосредственному 16-битному операнду так, чтобы получилось 32-битное число.

Ответ

1.

```
lw   $t0, 0x20($t7)
addi $s0, $0, -10
```
2.

```
0x8de80020    (lw)
0x2010fff6    (addi)
```

Упражнение 6.13

Условие

Повторите Упражнение 6.12 для инструкций из Упражнение 6.11.

Ответ

1.

```
addi $s0, $0, 73
sw   $t1, -7($t2)
```
2.

```
0x20100049    (addi)
0xad49fff9    (sw)
```


Упражнение 6.14

Условие

Преобразуйте следующую программу из машинного языка в программу на языке ассемблера MIPS. Цифрами слева показаны адреса инструкций в памяти. Цифры справа — это инструкции по соответствующим адресам. Объясните, что делает эта программа, предполагая, что перед началом её выполнения `$a0` содержит некое положительное число `n`, а после завершения программы в `$v0` получается некоторый результат. Также напишите эту программу на языке высокого уровня (например, C).

```
0x00400004 0x20090001
0x00400008 0x0089502A
0x0040000C 0x15400003
0x00400010 0x01094020
0x00400014 0x21290002
0x00400018 0x08100002
0x0040001C 0x01001020
0x00400020 0x03E00008
```

Ответ

Программа будет преобразована следующим образом:

```
        addi $t0, $0, 0
        addi $t1, $0, 1
loop:   slt  $t2, $a0, $t1
        bne $t2, $0, finish
        add $t0, $t0, $t1
        addi $t1, $t1, 2
        j   loop
finish: add $v0, $t0, $0
```

На C программу можно написать следующим образом(предполагая, что `temp = $t0`, `i = $t1`, `n = $a0`, `result = $v0`):

```
temp = 0;
for (i = 1; i <= n; i = i + 2)
    temp = temp + i;
result = temp;
```

Программа находит сумму нечётных чисел и кладёт результат в возвращаемый регистр `$v0`.

Упражнение 6.15

Условие

Повторите упражнение Упражнение 6.14 для приведенного ниже машинного кода. Используйте значения `$a0` и `$a1` как входные данные. В начале программы `$a0` содержит некоторое 32-битное число, а `$a1` — адрес некоторого массива из 32 символов (типа `char`).

```
0x00400000 0x2008001F
0x00400004 0x01044806
0x00400008 0x31290001
0x0040000C 0x0009482A
0x00400010 0xA0A90000
0x00400014 0x20A50001
0x00400018 0x2108FFFF
0x0040001C 0x0501FFF9
0x00400020 0x03E00008
```

Ответ

Программа будет преобразована следующим образом:

```
addi $t0, $0, 31
L1:
srlv $t1, $a0, $t0
andi $t1, $t1, 1
slt $t1, $0, $t1
sb $t1, 0($a1)
addi $a1, $a1, 1
addi $t0, $t0, -1
bgez $t0, L1
jr $ra
```

На C программу можно написать следующим образом:

```
void convert2bin(int num, char binarray[]) {
    int i;
    char tmp, val = 31;
    for (i = 0; i < 32; i++) {
        tmp = (num >> val) & 1;
        binarray[i] = tmp;
        val--;
    }
}
```

Данная программа переводит число из десятичной системы в двоичную и сохраняет результат в массив.

Упражнение 6.16

Условие

В архитектуре MIPS имеется инструкция `nor`, но отсутствует её вариант с непосредственным операндом `norl`. Тем не менее, команда `norl` может быть реализована существующими инструкциями. Напишите на языке ассемблера код со следующей функциональностью: `$t0 = $t1 NOR 0xF234`. Используйте наименьшее возможное число инструкций.

Ответ

```
ori $t0, $t1, 0xF234
nor $t0, $t0, $0
```

Упражнение 6.17

Условие

Реализуйте следующие фрагменты кода высокого уровня на языке ассемблера MIPS, используя инструкцию `slt`. Значения целочисленных переменных `g` и `h` хранятся в регистрах `$s0` и `$s1` соответственно.

1. `if (g > h)`
 `g = g + h;`
 `else`
 `g = g - h;`
2. `if (g >= h)`
 `g = g + 1;`
 `else`
 `h = h - 1;`
3. `if (g <= h)`
 `g = 0;`
 `else`
 `h = 0;`

Ответ

1.

```
# $s0 = g, $s1 = h
slt $t0, $s1, $s0      # if h < g, $t0 = 1
beq $t0, $0, else      # if $t0 == 0, do else
add $s0, $s0, $s1      # g = g + h
j done                 # jump past else block
else: sub $s0, $s0, $s1 # g = g - h
done:
```
2.

```
slt $t0, $s0, $s1      # if g < h, $t0 = 1
bne $t0, $0, else      # if $t0 != 0, do else
addi $s0, $s0, 1        # g = g + 1
j done                 # jump past else block
else: addi $s1, $s1, -1  # h = h - 1
done:
```

```

3. slt $t0, $s1, $s0      # if h < g, $t0 = 1
   bne $t0, $0, else      # if $t0 != 0, do else
   add $s0, $0, $0        # g = 0
   j    done              # jump past else block
else:  sub $s1, $0, $0    # h = 0
done:

```

Упражнение 6.18

Условие

Напишите функцию на языке высокого уровня (например, C), имеющую следующий прототип:

```
int find42(int array[], int size)
```

Здесь `array` задаёт базовый адрес некоторого массива целых чисел, а `size` содержит число элементов в этом массиве. Функция должна возвращать порядковый номер первого элемента массива, содержащего значение 42. Если в массиве нет числа 42, то функция должна вернуть -1.

Ответ

```

int find42(int array[], int size) {
    int i;
    for (i = 0; i < size; i = i + 1)
        if (array[i] == 42)
            return i;
    return -1;
}

```

Упражнение 6.19

Условие

Функция `strcpy` копирует строку символов, расположенную в памяти по адресу `src`, в новое место с адресом `dst`.

```
// C code
void strcpy(char dst[], char src[]) {
    int i = 0;
    do {
        dst[i] = src[i];
    } while (src[i++]);
}
```

1. Реализуйте приведенную выше функцию `strcpy` на языке ассемблера MIPS. Используйте `$s0` для `i`.
2. Изобразите стек до вызова, во время и после вызова функции `strcpy`. Считайте, что перед вызовом `strcpy` `$sp = 0x7FFFFFF0`.

Ответ

```
1. # MIPS assembly code
   # base address of array dst = $a0
   # base address of array src = $a1
   # i = $s0

strcpy:
    addi $sp, $sp, -4
    sw   $s0, 0($sp)    # save $s0 on the stack
    add  $s0, $0, $0    # i = 0

loop:
    add  $t1, $a1, $s0 # $t1 = address of src[i]
    lb   $t2, 0($t1)   # $t2 = src[i]
    add  $t3, $a0, $s0 # $t3 = address of dst[i]
    sb   $t2, 0($t3)   # dst[i] = src[i]
    beq  $t2, $0, done # check for null character
    addi $s0, $s0, 1    # i++
    j    loop

done:
    lw   $s0, 0($sp)    # restore $s0 from stack
    addi $sp, $sp, 4     # restore stack pointer
    jr   $ra# return
```

2. Стэк до(i), во время(ii) и после(iii) процедуры `strcpy`:

Address	Data		Address	Data		Address	Data	
0x7FFFFFF00		← \$sp	0x7FFFFFF00			0x7FFFFFF00		← \$sp
0x7FFFFFFFC			0x7FFFFFFFC	\$s0	← \$sp	0x7FFFFFFFC		
0x7FFFFFFE8			0x7FFFFFFE8			0x7FFFFFFE8		
0x7FFFFFFE4			0x7FFFFFFE4			0x7FFFFFFE4		
(i)			(ii)			(iii)		

Упражнение 6.20

Условие

Реализуйте функцию из Упражнение 6.18 на языке ассемблера MIPS.

Ответ

```
find42: addi $t0, $0, 0      # $t0 = i = 0
        addi $t1, $0, 42    # $t1 = 42
loop:   slt  $t3, $t0, $a1   # $t3 = 1 if i < size (not at end of array)
        beq  $t3, $0, exit   # if reached end of array, return -1
        sll  $t2, $t0, 2     # $t2 = i*4
        add  $t2, $t2, $a0   # $t2 = address of array[i]
        lw   $t2, 0($t2)     # $t2 = array[i]
        beq  $t2, $t1, done   # $t2 == 42?
        addi $t0, $t0, 1     # i = i + 1
        j    loop
done:   add  $v0, $t0, $0     # $v0 = i
        jr   $ra
exit:   addi $v0, $0, -1     # $v0 = -1
        jr   $ra
```

Упражнение 6.21

Условие

Рассмотрим приведенный ниже код на языке ассемблера MIPS. Функции `func1`, `func2` и `func3` — нелистовые (нетерминальные) функции, а `func4` — листовая (терминальная). Полный код функций не показан, но в комментариях указаны регистры, используемые каждой из них.

```
0x00401000    func1:...                # func1 uses $s0-$s1
0x00401020                jal func2
...
0x00401100    func2:...                # func2 uses $s2-$s7
0x0040117C                jal func3
...
0x00401400    func3:...                # func3 uses $s1-$s3
0x00401704                jal func4
...
0x00403008    func4:...                # func4 uses no preserved
0x00403118                jr $ra        # registers
```

1. Сколько слов занимает кадр стека у каждой из этих функций?
2. Изобразите стек после вызова `func4`. Укажите, какие регистры хранятся в стеке и где именно. Отметьте каждый из кадров стека. Там, где это возможно, подпишите значения, сохранённые в стеке.

Ответ

1. Кадр стека для каждой процедуры:

```
proc1: 3 words deep (for $s0 - $s1, $ra)
proc2: 7 words deep (for $s2 - $s7, $ra)
proc3: 4 words deep (for $s1 - $s3, $ra)
proc4: 0 words deep
```

2.

	Addresss	Data	
	7FFF FF00	\$ra	
Кадры стека proc1	7FFF FEFC	\$s0	
	7FFF FEF8	\$s1	
Кадры стека proc2	7FFF FEF4	\$ra = 0x00401024	
	7FFF FEF0	\$s2	
	7FFF FEEC	\$s3	
	7FFF FEE8	\$s4	
	7FFF FEE4	\$s5	
	7FFF FEE0	\$s6	
	7FFF FEDC	\$s7	
Кадры стека proc3	7FFF FEE8	\$ra = 0x00401180	
	7FFF FEE4	\$s1	
	7FFF FEE0	\$s2	
	7FFF FEDC	\$s3	← \$sp

Упражнение 6.22

Условие

Каждое число в последовательности Фибоначчи является суммой двух предыдущих чисел.

1. Чему равны значения `fib(n)` для `n = 0` и `n = -1`?
2. Напишите функцию с именем `fib` на языке высокого уровня (например, C). Функция должна возвращать число Фибоначчи для любого неотрицательного значения `n`. Прокомментируйте ваш код.
3. Преобразуйте функцию, написанную в части 2., в код на ассемблере MIPS. После каждой строки кода добавьте строку комментария, поясняющего, что она делает. Протестируйте код для случая `fib(9)` в симуляторе SPIM.

Ответ

```

1. fib(0) = 0
   fib(-1) = 1

2. int fib(int n) {
    int prevresult = 1; // fib(n-1)
    int result = 0;     // fib(n)
    while (n != 0) {    // вычисление нового значения
        result = result + prevresult; // fib(n) = fib(n-1) + fib(n-2)
        prevresult = result - prevresult; // fib(n-1) = fib(n) - fib(n-2)
        n = n - 1;
    }
    return result;
}

```



```

3. # Процедура fib() вычисляет n-ое число Фибоначчи.
   # n передаётся в fib() через $a0, и fib() возвращает результат в $v0.
main:  addi $a0,$0,9      # инициализация аргумента процедуры: n = 9
       jal  fib          # вызов процедуры fibonacsi
       ...              # остальной код
fib:   addi $t0,$0,1      # $t0 = fib(n-1) = fib(-1) = 1
       addi $t1,$0,0      # $t1 = fib(n) = fib(0) = 0
loop:  beq  $a0,$0, end   # конец?
       add  $t1,$t1,$t0    # Вычисление следующего значения Fib #, fib(n)
       sub  $t0,$t1,$t0    # Обновление fib(n-1)
       addi $a0,$a0,-1     # уменьшение n
       j    loop          # Повторение
end:   add  $v0,$t1,$0     # Передача значения в $v0
       jr   $ra           # возвращение результата

```

Упражнение 6.23

Условие

Обратимся к примеру кода ниже. Предположим, что функция `factorial` вызывается с аргументом `n = 5`.

```

// C code
int factorial(int n) {
    if (n <= 1)
        return 1;
    else
        return (n * factorial(n - 1));
}

# MIPS assembly code
factorial: addi $sp, $sp, -8 # make room on stack
           sw   $a0, 4($sp) # store $a0
           sw   $ra, 0($sp) # store $ra
           addi $t0, $0, 2   # $t0 = 2
           slt  $t0, $a0, $t0 # n <= 1 ?
           beq  $t0, $0, else # no: goto else
           addi $v0, $0, 1   # yes: return 1
           addi $sp, $sp, 8  # restore $sp
           jr   $ra         # return
else:      addi $a0, $a0, -1 # n = n - 1
           jal  factorial   # recursive call
           lw   $ra, 0($sp) # restore $ra
           lw   $a0, 4($sp) # restore $a0
           addi $sp, $sp, 8  # restore $sp
           mul  $v0, $a0, $v0 # n * factorial(n-1)
           jr   $ra         # return

```

1. Чему будет равен регистр `$v0`, когда функция `factorial` завершится и управление будет возвращено вызвавшей ее функции?
2. Предположим, что инструкции, сохраняющие и восстанавливающие `$ra`, расположенные по адресам `0x98` и `0xB8`, были убраны (например, заменены на `nop`). В этом случае программа (1) войдет в бесконечный цикл, но не завершится аварийно; (2) завершится аварийно (произойдет переполнение стека или счетчик команд выйдет за пределы программы); (3) вернет неправильное значение в `$v0` (если да, то какое?); (4) продолжит работать правильно, несмотря на изменения?
3. Повторите часть 2., когда будут удалены инструкции по следующим адресам: (i) `0x94` и `0xC0` (инструкции, которые сохраняют и восстанавливают `$a0`); (ii) `0x90` и `0xC4` (инструкции, которые сохраняют и восстанавливают `$sp`); (iii) `0xAC` (инструкция, которая восстанавливает `$sp`)

Ответ

1. 120
2. 2
3. (i) 3 — возвращаемое значение 1; (ii) 2; (iii) 4

Упражнение 6.24

Условие

Бен Битдидл попытался вычислить функцию $f(a,b) = 2a + 3b$ для положительного значения `b`, но переусердствовал с вызовами функций и рекурсией и написал вот такой код:

```
// high-level code for functions f and f2
int f(int a, int b) {
    int j;
    j = a;
    return j + a + f2(b);
}

int f2(int x) {
    int k;
    k = 3;
    if (x == 0)
        return 0;
    else
        return k + f2(x - 1);
}
```

После этого Бен транслировал эти две функции на язык ассемблера. Он также написал функцию `test`, которая вызывает функцию `f(5, 3)`.

```
# MIPS assembly code # f: $a0 = a, $a1 = b, $s0 = j; f2: $a0 = x, $s0 = k
test: addi $a0, $0, 5      # $a0 = 5 (a = 5)
      addi $a1, $0, 3      # $a1 = 3 (b = 3)
      jal  f               # call f(5, 3)
loop:  j    loop           # and loop forever
f:     addi $sp, $sp, -16   # make room on the stack
      # for $s0, $a0, $a1, and $ra
      sw   $a1, 12($sp)    # save $a1 (b)
      sw   $a0, 8($sp)     # save $a0 (a)
      sw   $ra, 4($sp)     # save $ra
      sw   $s0, 0($sp)     # save $s0
      add  $s0, $a0, $0     # $s0 = $a0 (j = a)
      add  $a0, $a1, $0     # place b as argument for f2
      jal  f2              # call f2(b)
      lw   $a0, 8($sp)     # restore $a0 (a) after call
      lw   $a1, 12($sp)    # restore $a1 (b) after call
      add  $v0, $v0, $s0    # $v0 = f2(b) + j
      add  $v0, $v0, $a0    # $v0 = (f2(b) + j) + a
      lw   $s0, 0($sp)     # restore $s0
      lw   $ra, 4($sp)     # restore $ra
      addi $sp, $sp, 16    # restore $sp (stack pointer)
      jr   $ra             # return to point of call
f2:    addi $sp, $sp, -12   # make room on the stack for
      # $s0, $a0, and $ra
      sw   $a0, 8($sp)     # save $a0 (x)
      sw   $ra, 4($sp)     # save return address
      sw   $s0, 0($sp)     # save $s0
      addi $s0, $0, 3      # k = 3
      bne  $a0, $0, else   # x = 0?
      addi $v0, $0, 0      # yes: return value should be 0
      j    done            # and clean up 0x00400070
else:  addi $a0, $a0, -1    # no: $a0 = $a0 - 1 (x = x - 1)
      jal  f2              # call f2(x - 1)
      lw   $a0, 8($sp)     # restore $a0 (x)
      add  $v0, $v0, $s0    # $v0 = f2(x - 1) + k0x00400080
done:  lw   $s0, 0($sp)     # restore $s0
      lw   $ra, 4($sp)     # restore $ra
      addi $sp, $sp, 12    # restore $sp
      jr   $ra             # return to point of call
```

1. Если код выполнится, начиная с метки `test`, то какое значение окажется в регистре `$v0`, когда программа дойдет до метки `loop`? Правильно ли программа вычислит $2a + 3b$?
2. Предположим, что Бен удалил инструкции по адресам `0x0040001C` и `0x00400044`, которые сохраняют и восстанавливают значение регистра `$ra`. В этом случае программа (1) войдет в бесконечный цикл, но не остановится; (2) завершится аварийно (произойдет переполнение стека или счетчик команд выйдет за пределы программы); (3) вернет неправильное значение в `$v0` (если да, то какое?); (4) будет работать правильно, несмотря на изменения?
3. Повторите часть 2., когда будут удалены инструкции по следующим адресам: (i) `0x00400018` и `0x00400030`; (ii) `0x00400014` и `0x00400034`; (iii) `0x00400020` и `0x00400040`; (iv) `0x00400050` и `0x00400088`; (vi) `0x00400058` и `0x00400084`; (vii) `0x00400054` и `0x00400078`.

Ответ

1. `$v0` выполнится как и задумано с результатом 19.
2. Программа завершится аварийно, поскольку стек переполнится.
3. (i) программа вернёт неправильное значение 17; (ii) будет работать правильно; (iii) будет работать правильно; (iv) войдет в бесконечный цикл; (v) программа вернёт неправильное значение 17; (vi) программа завершится аварийно, счётчик команд выйдет за пределы; (vii) будет работать правильно.

Упражнение 6.25

Условие

Переведите приведенные ниже инструкции `beq`, `j` и `jal` в машинный код. Адреса инструкций указаны слева от каждой из них:

```
(1)
0x00401000      beq $t0, $s1, Loop
0x00401004      ...
0x00401008      ...
0x0040100C  Loop: ...

(2)
0x00401000      beq $t7, $s4, done
...             ...
0x00402040  done: ...

(3)
0x0040310C  back: ...
...             ...
0x00405000      beq $t9, $s7, back

(4)
0x00403000      jal func
...             ...
0x0041147C  func: ...

(5)
0x00403004  back: ...
...             ...
0x0040400C  j      back
```

Ответ

1. 000100 01000 10001 0000 0000 0000 0010 = 0x11110002
2. 000100 01111 10100 0000 0100 0000 1111 = 0x11F4040F
3. 000100 11001 10111 1111 1000 0100 0010 = 0x1337F842
4. 000011 0000 0100 0001 0001 0100 0111 11 = 0x0C10451F
5. 000010 00 0001 0000 0000 1100 0000 0001 = 0x08100C01

Упражнение 6.26

Условие

Рассмотрим следующий фрагмент кода на языке ассемблера MIPS:

```
        add  $a0, $a1, $0
        jal  f2
f1:     jr   $ra
f2:     sw   $s0, 0($s2)
        bne  $a0, $0, else
        j    f1
else:   addi  $a0, $a0, -1
        j    f2
```

1. Транслируйте последовательность инструкций в машинный код в шестнадцатеричном формате.
2. Сделайте список режимов адресации, которые были использованы для каждой строки кода.

Ответ

1.

add	\$a0, \$a1, \$0	# 0x00a02020
jal	f2	# 0x0c10000d
f1:	jr	\$ra # 0x03e00008
f2:	sw	\$s0, 0(\$s2) # 0xae500000
	bne	\$a0, \$0, else # 0x14800001
	cj	f1 # 0x0810000c
else:	addi	\$a0, \$a0, -1 # 0x2084ffff
	j	f2 # 0x0810000d
2.

add	\$a0, \$a1, \$0	# register only
jal	f2	# pseudo-direct
f1:	jr	\$ra # register only
f2:	sw	\$s0, 0(\$s2) # base addressing
	bne	\$a0, \$0, else # PC-relative
	cj	f1 # pseudo-direct
else:	addi	\$a0, \$a0, -1 # immediate
	j	f2 # pseudo-direct

Упражнение 6.27

Условие

Рассмотрим следующий фрагмент кода на C:

```
void setArray(int num) {
    int i;
    int array[10];
    for (i = 0; i < 10; i = i + 1) {
        array[i] = compare(num, i);
    }
}

int compare(int a, int b) {
    if (sub(a, b) >= 0)
        return 1;
    else
        return 0;
}

int sub(int a, int b) {
    return a - b;
}
```

1. Перепишите этот фрагмент кода на языке ассемблера MIPS. Используйте регистр `$s0` для хранения переменной `i`. Массив хранится в стеке функции `setArray`.
2. Предположим, что первой вызванной функцией будет `setArray`. Нарисуйте состояние стека перед вызовом `setArray` и во время каждого последующего вызова. Укажите имена регистров и переменных, хранящихся в стеке. Отметьте расположение `$sp` и каждого кадра стека.
3. Как бы работал ваш код, если бы вы забыли сохранить в стеке регистр `$ra`?

Ответ

1.

```
set_array:  addi $sp,$sp,-52    # move stack pointer
            sw   $ra,48($sp)  # save return address
            sw   $s0,44($sp)  # save $s0
            sw   $s1,40($sp)  # save $s1

            add  $s0,$0,$0    # i = 0
            addi $s1,$0,10    # max iterations = 10
loop:      add  $a1,$s0,$0    # pass i as parameter
            jal  compare # call compare(num, i)
            sll  $t1,$s0,2    # $t1 = i*4
            add  $t2,$sp,$t1  # $t2 = address of array[i]
            sw   $v0,0($t2)   # array[i] = compare(num, i);
            addi $s0,$s0,1    # i++
            bne  $s0,$s1,loop # if i<10, goto loop
```

```

        lw    $s1,40($sp)    # restore $s1
        lw    $s0,44($sp)    # restore $s0
        lw    $ra,48($sp)    # restore return address
        addi  $sp,$sp,52     # restore stack pointer
        jr    $ra            # return to point of call

compare: addi  $sp,$sp,-4     # move stack pointer
        sw    $ra,0($sp)     # save return address on the stack
        jal   subtract        # input parameters already in $a0,$a1
        slt   $v0,$v0,$0      # $v0=1 if sub(a,b) < 0 (return 0)
        slti  $v0,$v0,1       # $v0=1 if sub(a,b)>=0, else $v0 = 0
        lw    $ra,0($sp)     # restore return address
        addi  $sp,$sp,4       # restore stack pointer
        jr    $ra            # return to point of call

subtract: sub   $v0,$a0,$a1    # return a-b
        jr    $ra            # return to point of call

```

2.

До setArray	Во время setArray	Во время compare/sub
\$sp →		
	\$ra	\$ra
	\$s0	\$s0
	\$s1	\$s1
	array[9]	array[9]
	array[8]	array[8]
	array[7]	array[7]
	array[6]	array[6]
	array[5]	array[5]
	array[4]	array[4]
	array[3]	array[3]
	array[2]	array[2]
	array[1]	array[1]
\$sp →	array[0]	array[0]
	\$sp →	\$ra

3. Если бы \$ra никогда не хранился в стеке, функция сравнения после вызова вернулась бы к инструкции вычитания (slt \$v0, \$v0, \$0) вместо возврата к функции setArray. Программа войдет в бесконечный цикл в функции сравнения между jr \$ra и slt \$v0, \$v0, \$0. Она будет увеличивать стек во время этого цикла до тех пор, пока пространство стека не будет превышено, и программа, скорее всего, выйдет из строя.

Упражнение 6.28

Условие

Рассмотрим следующий фрагмент кода на C:

```
int f(int n, int k) {
    int b;
    b = k + 2;
    if (n == 0)
        b = 10;
    else
        b = b + (n * n) + f(n - 1, k + 1);
    return b * k;
}
```

1. Транслируйте функцию `f` на язык ассемблера MIPS. Обратите особое внимание на правильность сохранения и восстановления регистров между вызовами функций, а также на использование конвенций MIPS по сохранению регистров. Тщательно комментируйте ваш код. Вы можете использовать инструкцию `mul`. Функция начинается с адреса `0x00400100`. Храните локальную переменную `b` в регистре `$s0`.
2. Пошагово выполните функцию из пункта 1. для случая `f(2, 4)`. Изобразите стек. Подпишите имена и значения регистров, хранящихся в каждом слове стека. Проследите за значением указателя стека (`$sp`). Четко обозначьте каждый кадр стека. Предположим, что при вызове `f` значение `$s0 = 0xABCD`, а `$ra = 0x400004`. Каким будет конечный результат в регистре `$v0`?

Ответ

```
1. # MIPS assembly code
f:   addi $sp, $sp, -16 # decrement stack
     sw   $a0, 0xc($sp) # save registers on stack
     sw   $a1, 0x8($sp)
     sw   $ra, 0x4($sp)
     sw   $s0, 0x0($sp)
     addi $s0, $a1, 2   # b = k + 2
     bne $a0, $0, else # if (n!=0) do else block
     addi $s0, $0, 10   # b = 100x400120
     j    done

else: addi $a0, $a0, -1 # update arguments
      addi $a1, $a1, 1
      jal  f          # call f()
      lw   $a0, 0xc($sp) # restore arguments
      lw   $a1, 0x8($sp)
      mult $a0, $a0      # {[hi],[lo]} = n*n
      mflo $t0           # $t0 = lo (assuming 32-bit result)
      add  $s0, $s0, $t0 # b = b + n*n
      add  $s0, $s0, $v0 # b = b + n*n + f(n-1,k+1)

done: mult $s0, $a1      # {[hi],[lo]} = b * k
```

```

mflo $v0          # $v0 = lo (assuming 32-bit result)
lw  $ra, 0x4($sp) # restore registers
lw  $s0, 0x0($sp)
addi $sp, $sp, 16  # restore stack pointer
jr  $ra           # return

```

2. Стек (i) после последнего рекурсивного вызова и (ii) после возврата значения. Финальное значение \$v0 будет 1400.

Address	Data		Address	Data	
.	.		.	.	
.	.	$\leftarrow \$sp$.	.	$\leftarrow \$sp; \$v0 = 1400$
7FFF FF00	\$a0 = 2		7FFF FF00	\$a0 = 2	
7FFF FEFC	\$a1 = 4		7FFF FEFC	\$a1 = 4	
7FFF FEF8	\$ra = 0x400004		7FFF FEF8	\$ra = 0x400004	
7FFF FEF4	\$s0 = 0xABCD	$\leftarrow \$sp$	7FFF FEF4	\$s0 = 0xABCD	$\leftarrow \$sp; \$s0 = 350; \$v0 = 1400$
7FFF FEF0	\$a0 = 1		7FFF FEF0	\$a0 = 1	
7FFF FEFC	\$a1 = 5		7FFF FEFC	\$a1 = 5	
7FFF FEF8	\$ra = 0x400130		7FFF FEF8	\$ra = 0x400130	
7FFF FEF4	\$s0 = 6	$\leftarrow \$sp$	7FFF FEF4	\$s0 = 6	$\leftarrow \$sp; \$s0 = 68; \$v0 = 340$
7FFF FEF0	\$a0 = 0		7FFF FEF0	\$a0 = 0	
7FFF FEFC	\$a1 = 6		7FFF FEFC	\$a1 = 6	
7FFF FEF8	\$ra = 0x400130		7FFF FEF8	\$ra = 0x400130	
7FFF FEF4	\$s0 = 7	$\leftarrow \$sp$	7FFF FEF4	\$s0 = 7	$\leftarrow \$sp; \$s0 = 10; \$v0 = 60$
.	.		.	.	
	(i)			(ii)	

Упражнение 6.29

Условие

Каков диапазон адресов, по которым инструкции ветвления, такие как `beq` и `bne`, могут выполнять переходы в MIPS? Дайте ответ в виде количества инструкций относительно адреса инструкции ветвления.

Ответ

От 32 К - 1 инструкций до ветвления и 32 К инструкций после ветвления.

Упражнение 6.30

Условие

Дайте ответы в виде количества инструкций относительно адреса инструкции безусловного перехода:

1. Как далеко вперед (то есть по направлению к большим адресам) может перейти команда безусловного перехода (j) в наихудшем случае? Наихудший случай — это когда переход не может быть осуществлен далеко. Объясните словами, используя по необходимости примеры.
2. Как далеко вперед может перейти команда безусловного перехода (j) в наилучшем случае? Наилучший случай — это когда переход может быть осуществлен дальше всего. Поясните ответ.
3. Как далеко назад (то есть по направлению к меньшим адресам) может перейти команда безусловного перехода (j) в наихудшем случае? Поясните ответ.
4. Как далеко назад может перейти команда безусловного перехода (j) в наилучшем случае? Поясните ответ.

Ответ

1. В худшем случае инструкция перехода может перейти только на одну инструкцию вперед. Например, следующий код невозможен. Приведенная ниже инструкция перехода (j loop) может выполнять переход только вперед на 0x0FFFFFFC:

```
0x0FFFFFFF8      j    loop
0x0FFFFFFFC
0x10000000  loop:  ...
```

2. В лучшем случае инструкция перехода может выполнить переход на 2^{26} инструкций вперед:

```
0x0FFFFFFFC      j    loop
0x10000000      ...
0x1FFFFFFFC  loop:  ...
```

3. В худшем случае инструкция перехода не может выполнить переход назад. Например, следующий код невозможен. Поскольку инструкция перехода добавляет четыре старших значащих бита PC + 4, эта инструкция перехода не может даже перейти к самой себе, не говоря уже о назад:

```
0x0FFFFFFFC  loop:  j    loop
0x10000000      ...
```

4. В лучшем случае инструкция перехода может выполнить переход назад максимум на $2^{26} - 2$ инструкций:

```
0x10000000  loop:  ...
...
0x1FFFFFFF8      j      loop
```

Упражнение 6.31

Условие

Объясните, почему выгодно иметь большое поле адреса (`addr`) в машинном формате команд безусловного перехода `j` и `jal`.

Ответ

Выгодно иметь большое поле адреса в машинном формате для команд перехода, чтобы увеличить диапазон адресов команд, к которым инструкция может перейти.

Упражнение 6.32

Условие

Напишите код на языке ассемблера, который переходит к инструкции, отстоящей на 64 мегаинструкции от начала этого кода. Предположим, что ваш код начинается с адреса `0x00400000`. Используйте минимальное количество инструкций.

Ответ

```
0x00400000    lui $t1, 0x1040
0x00400004    jr  $t1
```

Упражнение 6.33

Условие

Напишите функцию на языке высокого уровня, которая берет массив 32-разрядных целых чисел из 10 элементов, использующих прямой порядок следования байтов (от младшего к старшему, little-endian) и преобразует его в формат с обратным порядком (от старшего к младшему, big-endian). Перепишите код на языке ассемблера MIPS. Прокомментируйте весь ваш код и используйте минимальное количество инструкций.

Ответ

```
// high-level code
void little2big(int[] array) {
    int i;
    for (i = 0; i < 10; i = i + 1) {
        array[i] = ((array[i] << 24) |
                    ((array[i] & 0xFF00) << 8) |
                    ((array[i] & 0xFF0000) >> 8) |
                    ((array[i] >> 24) & 0xFF));
    }
}

# MIPS assembly code
# $a0 = base address of array
little2big:
    addi $t5, $0, 10 # $t5 = i = 10 (loop counter)
loop: lb  $t0, 0($a0) # $t0 = array[i] byte 0
    lb  $t1, 1($a0) # $t1 = array[i] byte 1
    lb  $t2, 2($a0) # $t2 = array[i] byte 2
    lb  $t3, 3($a0) # $t3 = array[i] byte 3
    sb  $t3, 0($a0) # array[i] byte 0 = previous byte 3
    sb  $t2, 1($a0) # array[i] byte 1 = previous byte 2
    sb  $t1, 2($a0) # array[i] byte 2 = previous byte 1
    sb  $t0, 3($a0) # array[i] byte 3 = previous byte 0
    addi $a0, $a0, 4 # increment index into array
    addi $t5, $t5, -1 # decrement loop counter
    beq  $t5, $0, done
    j    loop
done: jr  $ra
```

Упражнение 6.34

Условие

Рассмотрим две строки: `string1` и `string2`:

1. Напишите код на языке высокого уровня для функции под названием `concat`, которая соединяет их (склеивает их вместе):

```
void concat(char string1[], char string2[], char stringconcat[])
```

2. Перепишите код из части 1. на языке ассемблера MIPS.

Ответ

```
1. void concat(char[] string1, char[] string2, char[] stringconcat) {
    int i, j;
    i = 0;
    j = 0;

    while (string1[i] != 0) {
        stringconcat[i] = string1[i];
        i = i + 1;
    }
    while (string2[j] != 0) {
        stringconcat[i] = string2[j];
        i = i + 1;
        j = j + 1;
    }
    stringconcat[i] = 0;
}
```

```
2. concat:  lb    $t0, 0($a0)      # $t0 = string1[i]
            beq    $t0, $0, string2 # if end of string1, go to string2
            sb     $t0, 0($a2)      # stringconcat[i] = string1[i]
            addi   $a0, $a0, 1      # increment index into string1
            addi   $a2, $a2, 1      # increment index into stringconcat
            j      concat          # loop back
string2:    lb     $t0, 0($a1)      # $t0 = string2[j]
            beq    $t0, $0, done    # if end of string2, return
            sb     $t0, 0($a2)      # stringconcat[j] = string2[j]
            addi   $a1, $a1, 1      # increment index into string2
            addi   $a2, $a2, 1      # increment index into stringconcat
            j      string2
done:       sb     $0, 0($a2)       # append null to end of string
            jr     $ra
```

Упражнение 6.35

Условие

Напишите программу на ассемблере MIPS, которая складывает два положительных числа с плавающей точкой одинарной точности, которые хранятся в регистрах `$s0` и `$s1`. Не используйте специальные инструкции для работы с плавающей точкой. Продемонстрируйте, что ваш код работает надежно.

Ответ

```
# определение масок в сегменте глобальных данных
.data
mmask: .word 0x007FFFFF
emask: .word 0x7F800000
ibit: .word 0x00800000
obit: .word 0x01000000

.text

flpadd: lw $t4,mmask          # load mantissa mask
        and $t0,$s0,$t4      # extract mantissa from $s0 (a)
        and $t1,$s1,$t4      # extract mantissa from $s1 (b)
        lw $t4,ibit          # load implicit leading 1
        or $t0,$t0,$t4        # add the implicit leading 1 to mantissa
        or $t1,$t1,$t4        # add the implicit leading 1 to mantissa
        lw $t4,emask          # load exponent mask
        and $t2,$s0,$t4      # extract exponent from $s0 (a)
        srl $t2,$t2,23        # shift exponent right
        and $t3,$s1,$t4      # extract exponent from $s1 (b)
        srl $t3,$t3,23        # shift exponent right
match:  beq $t2,$t3,addsig     # check whether the exponents match
        bgeu $t2,$t3,shiftb    # determine which exponent is larger
shifta: sub $t4,$t3,$t2        # calculate difference in exponents
        srav $t0,$t0,$t4       # shift a by calculated difference
        add $t2,$t2,$t4        # update a's exponent
        j addsig               # skip to the add
shiftb: sub $t4,$t2,$t3        # calculate difference in exponents
        srav $t1,$t1,$t4       # shift b by calculated difference
        add $t3,$t3,$t4        # update b's exponent (not necessary)
addsig: add $t5,$t0,$t1        # add the mantissas
norm:   lw $t4,obit            # load mask for bit 24 (overflow bit)
        and $t4,$t5,$t4        # mask bit 24
        beq $t4,$0,done        # right shift not needed because bit 24=0
        srl $t5,$t5,1          # shift right once by 1 bit
        addi $t2,$t2,1         # increment exponent
        # continue on the next page
```

```

done:   lw $t4,mmask           # load mask
        and $t5,$t5,$t4       # mask mantissa
        sll $t2,$t2,23        # shift exponent into place
        lw $t4,emask          # load mask
        and $t2,$t2,$t4       # mask exponent
        or $v0,$t5,$t2        # place mantissa and exponent into $v0
        jr $ra                # return to caller

```

Упражнение 6.36

Условие

Покажите, как приведенная ниже программа MIPS загружается и выполняется в памяти.

```

main:
    addi $sp, $sp, -4
    sw   $ra, 0($sp)
    lw   $a0, x
    lw   $a1, y
    jal  diff
    lw   $ra, 0($sp)
    addi $sp, $sp, 4
    jr   $ra
diff:
    sub  $v0, $a0, $a1
    jr   $ra

```

1. Сначала отметьте рядом с каждой инструкцией ее адрес.
2. Нарисуйте таблицы символов для меток и их адресов.
3. Сконвертируйте все инструкции в машинный код.
4. Укажите размер сегментов данных (**data**) и кода (**text**) в байтах.
5. Нарисуйте карту памяти и укажите, где хранятся данные и команды.

Ответ

```

1. 0x00400000 main:  addi $sp, $sp, -4
    0x00400004 sw    $ra, 0($sp)
    0x00400008 lw    $a0, x
    0x0040000C lw    $a1, y
    0x00400010 jal   diff
    0x00400014 lw    $ra, 0($sp)
    0x00400018 addi  $sp, $sp, 4
    0x0040001C jr    $ra

    0x00400020 diff:  sub  $v0, $a0, $a1
    0x00400024 jr    $ra

```


2.

symbol	address
x	0x10000000
y	0x10000004
main	0x00400000
diff	0x00400020

3.

Executable file header	Text Size	Data Size	
	0x28 (40 bytes)	0x8 (8 bytes)	
Text segment	Address	Instruction	
	0x00400000	0x23BDFFFC	addi \$sp, \$sp, -4
	0x00400004	0xAFBF0000	sw \$ra, 0(\$sp)
	0x00400008	0x8F848000	lw \$a0, 0x8000(\$gp)
	0x0040000C	0x8F858004	lw \$a1, 0x8004(\$gp)
	0x00400010	0x0C100008	jal diff
	0x00400014	0x8FBF0000	lw \$ra, 0(\$sp)
	0x00400018	0x23BD0004	addi \$sp, \$sp, 4
	0x0040001C	0x03E00008	jr \$ra
	0x00400020	0x00851022	sub \$v0, \$a0, \$a1
	0x00400024	0x03E00008	jr \$ra
Data segment	Address	Data	
	0x10000000	x	
	0x10000004	y	

4. Сегмент данных составляет 8 байтов, а текстовый сегмент — 40 (0x28) байтов.

5.

Address	Memory	
	Reserved	
0x7FFFFFFC	Stack	$\leftarrow \$sp = 0x7FFFFFFC$
	↓	
	↑	
0x10010000	Heap	
	.	
	.	$\leftarrow \$gp = 0x10008000$
	.	
	x	
0x10000000	y	
	.	
	.	
	.	
	0x03E00008	
	0x00851022	
	0x03E00008	
	0x23BD0004	
	0x8FBF0000	
	0x0C100008	
	0x8F858004	
	0x8F848000	
	0xAFBF0000	
0x00400000	0x23BDFFFC	$\leftarrow PC = 0x00400000$
	Reserved	

Упражнение 6.37

Условие

Повторите Упражнение 6.36 для следующего кода:

```
# MIPS assembly code
main:
    addi    $sp, $sp, -4
    sw      $ra, 0($sp)
    addi    $t0, $0, 15
    sw      $t0, a
    addi    $a1, $0, 27
    sw      $a1, b
    lw      $a0, a
    jal     greater
    lw      $ra, 0($sp)
    addi    $sp, $sp, 4
    jr      $ra
greater:
    slt     $v0, $a1, $a0
    jr      $ra
```

Ответ

```
1. 0x00400000 main:    addi $sp, $sp, -4
   0x00400004 sw      $ra, 0($sp)
   0x00400008 addi    $t0, $0, 15
   0x0040000C sw      $t0, 0x8000($gp)
   0x00400010 addi    $a1, $0, 27
   0x00400014 sw      $a1, 0x8004($gp)
   0x00400018 lw      $a0, 0x8000($gp)
   0x0040001C jal     greater
   0x00400020 lw      $ra, 0($sp)
   0x00400024 addi    $sp, $sp, 4
   0x00400028 jr      $ra

   0x0040002C greater: slt    $v0, $a1, $a0
   0x00400030          jr      $ra
```

2.

symbol	address
a	0x10000000
b	0x10000004
main	0x00400000
greater	0x00400020

3.

Executable file header	Text Size	Data Size	
	0x34 (52 bytes)	0x8 (8 bytes)	
Text segment	Address	Instruction	
	0x00400000	0x23BDFFFC	<code>addi \$sp, \$sp, -4</code>
	0x00400004	0xAFBF0000	<code>sw \$ra, 0(\$sp)</code>
	0x00400008	0x2008000F	<code>addi \$t0, \$0, 15</code>
	0x0040000C	0xAF888000	<code>sw \$t0, 0x8000(\$gp)</code>
	0x00400010	0x2005001B	<code>addi \$a1, \$0, 27</code>
	0x00400014	0xAF858004	<code>sw \$a1, 0x8004(\$gp)</code>
	0x00400018	0x8F848000	<code>lw \$a0, 0x8000(\$gp)</code>
	0x0040001C	0x0C10000B	<code>jal greater</code>
	0x00400020	0x8FBF0000	<code>lw \$ra, 0(\$sp)</code>
	0x00400024	0x23BD0004	<code>addi \$sp, \$sp, 4</code>
	0x00400028	0x03E00008	<code>jr \$ra</code>
	0x0040002C	0x00A4102A	<code>slt \$v0, \$a1, \$a0</code>
	0x00400030	0x03E00008	<code>jr \$ra</code>
Data segment	Address	Data	
	0x10000000	a	
	0x10000004	b	

4. Сегмент данных составляет 8 байтов, а текстовый сегмент — 52 (0x34) байтов.

5.

Address	Memory	
	Reserved	
0x7FFFFFFC	Stack	$\leftarrow \$sp = 0x7FFFFFFC$
	↓	
	↑	
0x10010000	Heap	
	.	
	.	$\leftarrow \$gp = 0x10008000$
	.	
	b	
0x10000000	a	
	.	
	.	
	.	
	0x03E00008	
	0x00A4102A	
	0x03E00008	
	0x23BD0004	
	0x8FBF0000	
	0x0C10000B	
	0x8f848000	
	0xAF850004	
	0x2005001B	
	0xAF888000	
	0x2008000F	
	0xAFBF0000	
0x00400000	0x23BDFFFC	$\leftarrow PC = 0x00400000$
	Reserved	

Упражнение 6.38

Условие

Покажите инструкции MIPS, которые реализуют следующие псевдокоманды. Вы можете использовать регистр `$at` для хранения временных данных, но вам запрещается портить содержимое (затирать) другие регистры.

1. `addi $t0, $s2, imm31:0.`
2. `lw $t5, imm31:0($s0).`
3. `rol $t0, $t1, 5.`
4. `ror $s4, $t6, 31.`

Ответ

1. `addi $t0, $2, imm31:0`

```
lui $at, imm31:16
ori $at, $at, imm15:0
add $t0, $2, $at
```

2. `lw $t5, imm31:0($s0)`

```
lui $at, imm31:16
ori $at, $at, imm15:0
add $at, $at, $s0
lw $t5, 0($at)
```

3. `rol $t0, $t1, 5`

```
srl $at, $t1, 27
sll $t0, $t1, 5
or $t0, $t0, $at
```

4. `ror $s4, $t6, 31`

```
sll $at, $t6, 1
srl $s4, $t6, 31
or $s4, $s4, $at
```

Упражнение 6.39

Условие

Повторите Упражнение 6.38 для следующих псевдокоманд:

1. `beq $t1, imm31:0, L`.
2. `ble $t3, $t5, L`.
3. `bgt $t3, $t5, L`.
4. `bge $t3, $t5, L`.

Ответ

1. `beq $t1, imm31:0, L`

`lui $at, imm31:16`
`ori $at, $at, imm15:0`
`beq $t1, $at, L`
2. `ble $t3, $t5, L`

`slt $at, $t5, $t3`
`beq $at, $0, L`
3. `bgt $t3, $t5, L`

`slt $at, $t5, $t3`
`bne $at, $0, L`
4. `bge $t3, $t5, L`

`slt $at, $t3, $t5`
`beq $at, $0, L`