Архитектура вычислительных систем

Лекция 7.

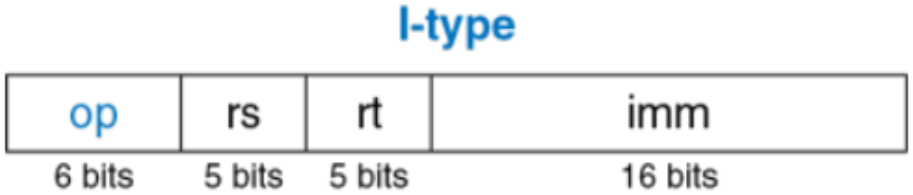
Яревский Е.А.

Кафедра вычислительной физики

Форматы команд

Инструкции типа I
от непосредственного типа (immediate-type).
Используют в качестве операндов два регистра и один непосредственный операнд (константу).

Команда состоит из четырёх полей: *op*, *rs*, *rt* и *imm*.
Первые три поля (*op*, *rs* и *rt*) аналогичны полям в командах типа **R**.
Поле *imm* содержит 16-битную константу.



Операция закодирована полем *op*.
Операнды заданы в трёх полях: *rs*, *rt* и *imm*.
Поля *rs* и *imm* всегда используются как операнды-источники.
Поле *rt* в некоторых командах содержит номер регистра-назначения, в других – номер регистра-источника.

Порядок операндов в ассемблерной записи и машинной команде **может быть разным!**

Форматы команд

Как 16-битные константы расширяются до 32 бит?
У неотрицательных констант верхние 16 бит будут заполнены нулями, а у отрицательных констант они будут заполнены единицами. (*расширение знака*). Значение не меняется!

Большинство команд производят расширение знака у непосредственных операндов.
Исключения – логические операции (*andi*, *ori*, *xori*), которые вместо расширения знака делают дополнение нулями.

Пример кодировки команд:

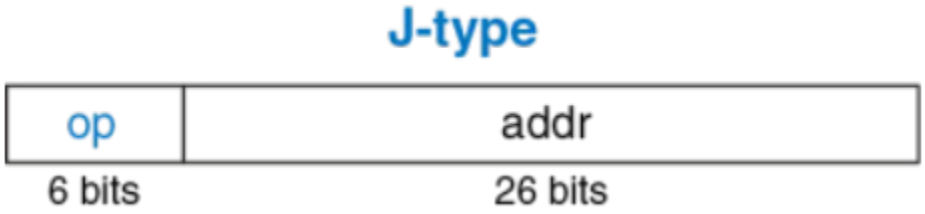
Assembly Code	Field Values				Machine Code				
	op	rs	rt	imm	op	rs	rt	imm	
`addi $s0, $s1, 5`	8	17	16	5	001000	10001	10000	0000 0000 0000 0101	(0x22300005)
`addi $t0, $s3, -12`	8	19	8	–12	001000	10011	01000	1111 1111 1111 0100	(0x2268FFF4)
`lw $t2, 32($0)`	35	0	10	32	100011	00000	01010	0000 0000 0010 0000	(0x8C0A0020)
`sw $s1, 4($t1)`	43	9	17	4	101011	01001	10001	0000 0000 0000 0100	(0xAD310004)
	6 bits	5 bits	5 bits	16 bits	6 bits	5 bits	5 bits	16 bits	

Форматы команд

Инструкции типа J

Этот формат используется только для инструкций безусловного перехода и ветвления.

В формате команд этого типа определён один 26-битный операнд *addr*.



Команды типа J начинаются с 6-битного поля кода операции (*opcode*). Оставшиеся биты используются для указания адреса перехода (*addr*).

Примеры инструкций

Арифметические/логические инструкции

Source Registers

$s1	1111	1111	1111	1111	0000	0000	0000	0000
$s2	0100	0110	1010	0001	1111	0000	1011	0111

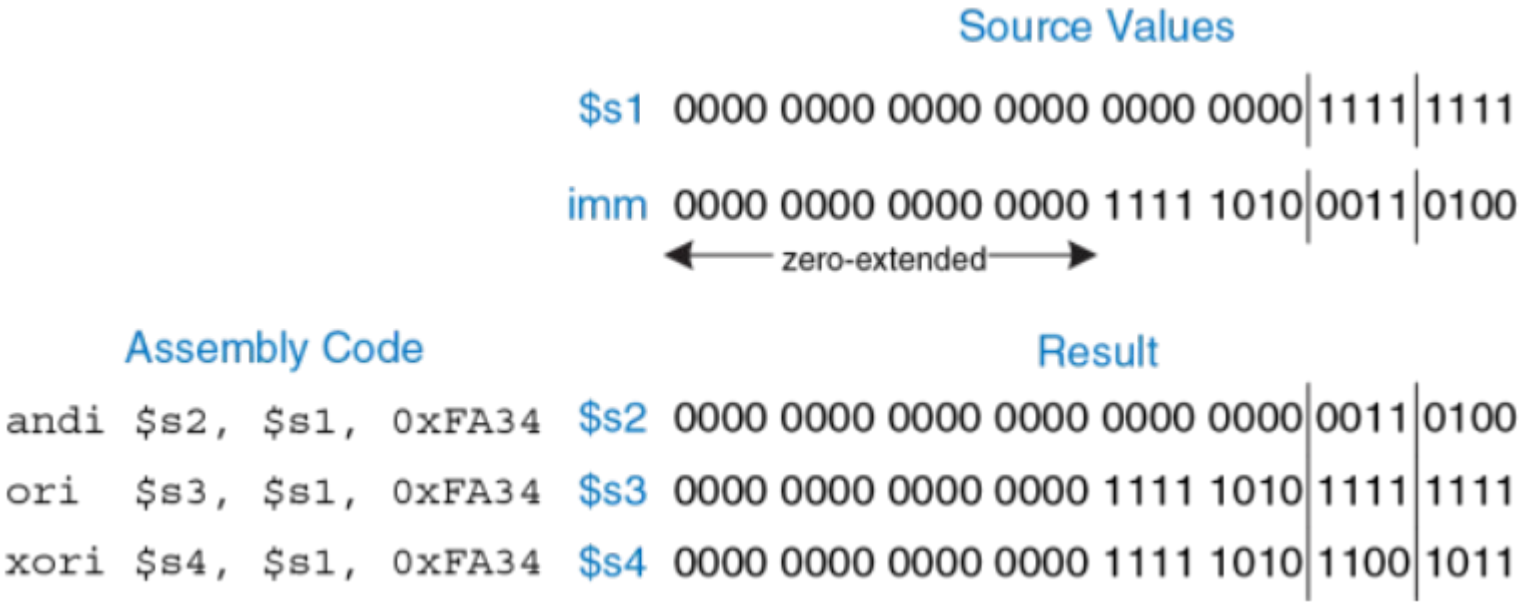
Assembly Code

```
and $s3, $s1, $s2
or  $s4, $s1, $s2
xor $s5, $s1, $s2
nor $s6, $s1, $s2
```

Result

$s3	0100	0110	1010	0001	0000	0000	0000	0000
$s4	1111	1111	1111	1111	1111	0000	1011	0111
$s5	1011	1001	0101	1110	1111	0000	1011	0111
$s6	0000	0000	0000	0000	0000	1111	0100	1000

Инструкции типа R, побитовые операции.



Логические инструкции типа I, побитовые операции.

Инструкции сдвига сдвигают значение в регистре влево/вправо на любое заданное количество бит, вплоть до 31.
Операции сдвига умножают/делят сдвигаемые значения на степени двойки.

В MIPS имеются следующие инструкции сдвига:
sll (логический сдвиг влево, *shift left logical*),
srl (логический сдвиг вправо, *shift right logical*) и
sra (арифметический сдвиг вправо, *shift right arithmetic*).

Сдвиги влево всегда заполняют освобождающиеся младшие биты нулями.
Сдвиги вправо могут быть как
логическими (в освобождающиеся старшие биты задвигаются нули), так и **арифметическими** (освобождающиеся старшие биты заполняются значением знакового бита).

Assembly Code	Field Values						Machine Code						
	op	rs	rt	rd	shamt	funct	op	rs	rt	rd	shamt	funct	
`sll $t0, $s1, 4`	0	0	17	8	4	0	000000	00000	10001	01000	00100	000000	(0x00114100)
`srl $s2, $s1, 4`	0	0	17	18	4	2	000000	00000	10001	10010	00100	000010	(0x00119102)
`sra $s3, $s1, 4`	0	0	17	19	4	3	000000	00000	10001	10011	00100	000011	(0x00119903)
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

	Source Values							
$s1	1111	0011	0000	0000	0000	0010	1010	1000
shamt								00100

Assembly Code		Result							
sll $t0, $s1, 4	$t0	0011	0000	0000	0000	0010	1010	1000	0000
srl $s2, $s1, 4	$s2	0000	1111	0011	0000	0000	0000	0010	1010
sra $s3, $s1, 4	$s3	1111	1111	0011	0000	0000	0000	0010	1010

Результат действия команд сдвига.

Инструкции переменного сдвига:
sllv (логический переменный сдвиг влево, *shift left logical variable*),
srlv (логический переменный сдвиг вправо, *shift right logical variable*) и
srav (арифметический переменный сдвиг вправо, *shift right arithmetic variable*).

Assembly Code	Field Values						Machine Code						
	op	rs	rt	rd	shamt	funct	op	rs	rt	rd	shamt	funct	
`sllv $s3, $s1, $s2`	0	18	17	19	0	4	000000	10010	10001	10011	00000	000100	(0x02519804)
`srlv $s4, $s1, $s2`	0	18	17	20	0	6	000000	10010	10001	10100	00000	000110	(0x0251A006)
`srav $s5, $s1, $s2`	0	18	17	21	0	7	000000	10010	10001	10101	00000	000111	(0x0251A807)
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Ассемблерные инструкции переменного сдвига имеют форму
sllv rd, rt, rs.
Операнды *rt* и *rs* следуют в обратном порядке по сравнению с большинством инструкций типа R.
Регистр *rt* содержит сдвигаемое значение, а пять младших бит поля *rs* определяют величину сдвига.
Результат сдвига помещается в регистр *rd*.
Поле *shamt* не используется и должно быть равно нулю.

	Source Values							
$s1	1111	0011	0000	0100	0000	0010	1010	1000
$s2	0000	0000	0000	0000	0000	0000	0000	**1000**

Assembly Code		Result							
`sllv $s3, $s1, $s2`	$s3	0000	0100	0000	0010	1010	1000	0000	0000
`srlv $s4, $s1, $s2`	$s4	0000	0000	1111	0011	0000	0100	0000	0010
`srav $s5, $s1, $s2`	$s5	1111	1111	1111	0011	0000	0100	0000	0010

Результат действия команд переменного сдвига.

Для присвоения переменным значений 16-битных констант используется addi:

```
# $s0 = a
addi $s0, $0, 0x4f3c # a = 0x4f3c
```

Для присвоения переменным значений 32-битных констант используется ***lui*** (***load upper immediate***), которая загружает константу в старшие 16 бит регистра и обнуляет младшие 16 битов,
а также инструкцию ***ori*** для загрузки константы в младшие 16 бит без изменения старших.

```
# $s0 = a
lui $s0, 0x6d5e # a = 0x6d5e0000
ori $s0, $s0, 0x4f3c # a = 0x6d5e4f3c
```

Умножение двух 32-битных чисел даёт 64-битное произведение.
Деление двух 32-битных чисел даёт 32-битное частное и 32-битный остаток.

В MIPS определено два регистра специального назначения **hi** и **lo**,
в которые сохраняются результаты умножения и деления.
Инструкция
mult $s0, $s1
умножает значения из регистров $s0 и $s1.
Старшие 32 бита произведения помещаются в регистр *hi*, а младшие – в регистр *lo*.
Аналогично, инструкция
div $s0, $s1
вычисляет значение $s0/$s1.
Частное помещается в *lo*, а остаток – в *hi*.

Инструкция ***mfhi $s2*** (пересылка из регистра hi, move from hi)
копирует значение из регистра hi в $s2.
Инструкция ***mflo $s3*** (пересылка из регистра lo, move from lo) копирует
значение из регистра lo в $s3.

Инструкция
mul $s1, $s2, $s3
умножает значения из $s2 и $s3 и сохраняет 32-битный результат в $s1
(при использовании инструкции mul старшие 32 бита произведения нигде
не сохраняются).

Для последовательного выполнения инструкций счетчик команд увеличивается на 4 после каждой из них.
Для изменения этого порядка нужны специальные команды, условные и безусловные.

Две основные инструкции условного перехода:
ветвление при равенстве (***beq***, *branch if equal*) и
ветвление при неравенстве (***bne***, *branch if not equal*).
Инструкция *beq* осуществляет переход, когда содержимое двух регистров равно, а *bne* осуществляет переход, если оно не равно.

Пример кода для *beq*:

```
    addi $s0, $0, 4          # $s0 = 0 + 4 = 4
    addi $s1, $0, 1          # $s1 = 0 + 1 = 1
    sll  $s1, $s1, 2         # $s1 = 1 << 2 = 4
    beq  $s0, $s1, target    # $s0 = = $s1, so branch is taken
    addi $s1, $s1, 1         # not executed
    sub  $s1, $s1, $s0       # not executed
target:
    add $s1, $s1, $s0 # $s1 = 4 + 4 = 8
```

Пример кода для *bne*:

```
    addi $s0, $0, 4          # $s0 = 0 + 4 = 4
    addi $s1, $0, 1          # $s1 = 0 + 1 = 1
    s11 $s1, $s1, 2          # $s1 = 1 << 2 = 4
    bne $s0, $s1, target     # $s0 = = $s1, so branch is not taken
    addi $s1, $s1, 1         # $s1 = 4 + 1 = 5
    sub $s1, $s1, $s0        # $s1 = 5 − 4 = 1
target:
    add $s1, $s1, $s0        # $s1 = 1 + 4 = 5
```

Три типа:
j – обычный безусловный переход (jump),
jal – безусловный переход с возвратом (jump and link) и
jr – безусловный переход по регистру (jump register).

Безусловный переход (***j***) осуществляет переход к инструкции по указанной метке.
Безусловный переход с возвратом (***jal***) похож на ***j***, но дополнительно сохраняет адрес возврата и используется при вызове функций.
Безусловный переход по регистру (***jr***) осуществляет переход к инструкции, адрес которой хранится в одном из регистров процессора.

Инструкции ***j*** и ***jal*** являются инструкциями типа **J**.
Инструкция ***jr*** является инструкцией типа **R**, но использует только операнд *rs*.

Пример кода:

```
    addi $s0, $0, 4        # $s0 = 4
    addi $s1, $0, 1        # $s1 = 1
    j target               # jump to target
    addi $s1, $s1, 1       # not executed
    sub $s1, $s1, $s0      # not executed
target:
    add $s1, $s1, $s0      # $s1 = 1 + 4 = 5
```

Пример кода:

```
0x00002000  addi $s0,$0, 0x2010   # $s0 = 0x2010
0x00002004  jr $s0                # jump to 0x00002010
0x00002008  addi $s1,$0, 1        # not executed
0x0000200c  sra $s1,$s1, 2        # not executed
0x00002010  lw $s3,44($s1)        # executed after jr instruction
```

Пример реализации оператора **if**

Код на языке высокого уровня

```
    if (i == j)
        f = g + h;
    f = f – i;
```

Код на языке ассемблера MIPS

```
    # $s0 = f, $s1 = g, $s2 = h, $s3 = i, $s4 = j
    bne $s3, $s4, L1      # if i != j, skip if block
    add $s0, $s1, $s2     # if block: f = g + h
L1:
    sub $s0, $s0, $s3     # f = f – i
```

Код на языке высокого уровня

```
int sum = 0;
for (i = 0; i != 10; i = i + 1) {
        sum = sum + i ;
}
```

Код на языке ассемблера MIPS

```
#       $s0 = i, $s1 = sum
        add $s1, $0, $0         # sum = 0
        addi $s0, $0, 0         # i = 0
        addi $t0, $0, 10        # $t0 = 10
for:
        beq $s0, $t0, done      # if i = = 10, branch to done
        add $s1, $s1, $s0       # sum = sum + i
        addi $s0, $s0, 1        # increment i
        j for
done:
```