**Оглавление.**

[Архитектура MIPS. 2](#_Toc58499402)

[Краткая справка. 2](#_Toc58499403)

[Формат инструкций. 2](#_Toc58499404)

[Язык ассемблера MIPS. 3](#_Toc58499405)

[Архитектура x86. 7](#_Toc58499406)

[Краткая справка. 7](#_Toc58499407)

[Формат инструкций. 7](#_Toc58499408)

[Язык ассемблера x86. 8](#_Toc58499409)

# Архитектура MIPS.

## Краткая справка.

**MIPS** – система команд и микропроцессорных архитектур, разработанных компанией MIPS Computer Systems. Ранние модели процессора имели 32-битную структуру, позже появились 64-битные версии.

На данный момент существует множество модификаций процессора, но действующими из них является только MIPS32 и MIPS64.

Таблица 1. MIPS

|  |  |
| --- | --- |
| MIPS | |
| Разработчик: | MIPS Technologies, Inc |
| Разрядность: | 32, 64 |
| Представлена: | 1985 |
| Архитектура: | RISC |
| Тип: | Регистр-регистр |
| Кодирование СК: | Фиксированное |
| Реализация переходов: | По сравнению двух регистров |
| Порядок байтов: | Bi-endian |
| Регистры | |
| Общего назначения: | 31 |
| Вещественные: | 32 |

## Формат инструкций.

В MIPS есть три типа инструкций: R, J и I.

Инструкции R-типа – работают только со значениями регистров.

Инструкции I-типа – работают со значениями регистров и используют константу, явно заданную в инструкции.

Инструкции J-типа – инструкции безусловного перехода; используют только константу из самой инструкции.

Каждая инструкция начинается с 6-битного кода. В дополнении к коду, инструкции R-типа определяют три регистра, область размера сдвига регистра и область функции; инструкции I-типа определяют два регистра и непосредственное значение; инструкции J-типа состоят из кода операции и 26-битного адреса перехода.

В таблице 2 подробно описана составляющая каждой из инструкции:

Таблица 2. Формат инструкций архитектуры MIPS.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Тип | Формат (в битах) | | | | | |
| R | Opcode (6) | Rs(5) | Rt(5) | Rd(5) | Shamt(5) | Функция(6) |
| I | Opcode (6) | Rs(5) | Rt(5) | Значение константы | | |
| J | Opcode (6) | Адрес безусловного перехода | | | | |

* opcode – код операции (показывает какого типа операция)
* rs, rt – номера регистров, значение которых используется
* rd – номер регистра, в который пишется значение
* shamt/shift – хранит значение, на которое нужно сдвинуть массив бит регистра
* funct – код конкретной операции R-типа

## Язык ассемблера MIPS.

В таблицах 3 и 4 показаны инструкции языка, которые имеют прямую аппаратную реализацию.

Таблица 3. Целочисленные операции языка ассемблера MIPS.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Название | Синтаксис  инструкции | Значение | Формат/  код/  функция | | | Примечание |
| Add | Add $d, $s, $t | $d=$s+$t | R | 0 | 20 | Складывает два регистра, выполняет прерывание при переполнении. |
| Add  unsigned | Addu $d, $s, $t | $d=$s+$t | R | 0 | 21 | Складывает два регистра, игнорирует переполнение. |
| Subtract | Sub $d, $s, $t | $d=$s-$t | R | 0 | 22 | Вычитает два регистра, выполняет прерывание при переполнении. |
| Subtract unsigned | Subu $d, $s, $t | $d=$s-$t | R | 0 | 22 | Вычитает два регистра, игнорирует переполнение. |
| Add immediate | Addi $t, $s, C | $t=$s + C | I | 8 |  | Используется для сложения констант со знаками, выполняет прерывание при переполнении. |
| Add immediate  unsigned | Addiu $t, $s, C | $t=$s+C | I | 9 |  | Используется для сложения констант со знаками, игнорирует переполнение. |
| Multiply | Mult $s, $t | LO=(($s\*$t)<<32)>>32  HI=($s\*$t)>>32 | R | 0 | 18 | Умножает два регистра и записывает 64-битный результат в два специальных поля для памяти – LO и HI. |
| Divide | Div $s, $t | LO=$s/$t  HI=$s % $t | R | 0 | 1A | Делит один регистра на другой и записывает 32-битный результат в LO, а остаток в HI |
| Divide unsigned | Divu $s, $t | LO=$s/$t  HI=$s % $t | R | 0 | 1B | Делит один регистр на другой и записывает 32-битный результат в LO, а остаток -–в HI. |

Таблица 4. Операции передачи данных языка ассемблера MIPS.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Название | Синтаксис  инструкции | Значение | Формат/  код/  функция | | | Примечание |
| Load double word | Ld $t, C($s) | $t=Memory[$s+C] | I | 23 | - | Загружает double word из MEM[$s+C] и следующих 7 байтов в $t и следующий регистр. |
| Load word | Lw $t, C($s) | $t=Memory[$s+C] | I | 22 | - | Загружает double word из MEM[$s+C] и следующих 3 байтов. |
| Load halfword | Lh $t, C($s) | $t=Memory[$s+C] (знаковое) | I | 21 | - | Загружает double word из MEM[$s+C] и следующего байта. Знак расширен до ширины регистра. |
| Load halfword unsigned | Lhu $t, C($s) | $t=Memory[$s+C] (беззнаковое) | I | 25 | - | Загружает double word из MEM[$s+C] и следующего байта. Без расширения знака. |
| Load byte | Lb $t, C($s) | $t=Memory[$s+C] (знаковое) | I | 20 | - | Загружает байт из MEM[$s+C].  Знак расширен до ширины регистра. |
| Load byte unsigned | Lbu $t, C($s) | $t=Memory[$s+C] (беззнаковое) | I | 24 | - | Загружает байт из MEM[$s+C].  Без расширения знака. |
| Store double word | sb $s, С($t) | Memory[$s+C]=$t | I | - | - | Сохраняет два типа word из $t и следующего регистра в MEM[$s+C] и следующих 7 байтах. |
| Store word | sw $s, С($t) | Memory[$s+C]=$t | I | 2B | - | Сохраняет тип word из $t в MEM[$s+C] и следующих 3 байтах. |
| Store half | sh $s, С($t) | Memory[$s+C]=$t | I | 29 | - | Сохраняет тип half word из $t (первую половину регистра) в MEM[$s+C] и следующем байте. |
| Store byte | sb $s, С($t) | Memory[$s+C]=$t | I | 28 | - | Сохраняет тип byte из $t (первую четверть регистра) в MEM[$s+C]. |
| Store half | sh $s, С($t) | Memory[$s+C]=$t | I | 29 | - | Сохраняет тип half word из $t в MEM[$s+C] и следующем байте. |
| Load upper immediate | Lui $t, C | $t=C<<16 | I | F |  | Загружает 16-битный операнд в вышестоящие 16 битов определенного регистра. Максимальная величина |
| Move from high | Mfhi $d | $d=HI | R | 0 | 10 | Помещает значение из HI в регистр. |
| Move from low | Mflo $d | $d=LO | R | 0 | 12 | Помещает значение из LO в регистр. |
| Move from Control Register | mfcZ $t, $s | $t=Coprocessor[Z].ControlRegister[$s] | R | 0 |  | Перемещает 4-байтовое значение из сопроцессора регистра Z-контроля в регистр общего назначения. Расширение знака. |
| Move to Control Register | mtcZ $t, $s | Coprocessor[Z].ControlRegister[$s]=$t | R | 0 |  | Перемещает 4-байтовое значение из регистра общего назначения в сопроцессор регистра Z-контроля в регистр общего назначения. Расширение знака. |

# Архитектура x86.

## Краткая справка.

**x86 –** архитектура процессора и одноименный набор команд, впервые реализованные в процессорах компании Intel. Поддерживается процессорами: AMD, VIA, Transmeta и т.д.

Таблица 5. x86

|  |  |
| --- | --- |
| MIPS | |
| Разработчик: | Intel, AMD |
| Разрядность: | 16, 32, 64 |
| Представлена: | 1978 |
| Архитектура: | CISC |
| Тип: | Регистр-память |
| Кодирование СК: | Переменной длины (1-15 байт) |
| Реализация переходов: | Флаги условий |
| Порядок байтов: | little-endian |
| Регистры | |
| Общего назначения: | 16-битные: 6 регистров + BP и SP  32-битные: 6 регистров процессора + EBP и ESP  64-битные: 14 регистров процессора + RBP и RSP |
| Вещественные: | 8\*80 бит (стековая организация) |
| SIMD (одиночный поток команд и множество потоков данных): | X86-64: 16\*128 бит  SSE: 8\*128 бит |

## Формат инструкций.

Аналогично MIPS, в архитектуре x86 есть 3 типа инструкций: R, J и I. Но

в отличие от MIPS, формат инструкций x86 устроен намного сложнее. Так, например, помимо точно определяющих инструкцию байтов, таких как opcode, params, modrm и sib, инструкция также включает в себя префиксы и встроенные в инструкцию данные. Более подробное описание дано в таблице 3.

Таблица 6. Формат инструкций архитектуры x86.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REX | Opcode | Mod\_R/M | SIB | Displacement | Immediate |
| 0-1 байт | 1-3 байта | 0-1 байт | 0-1 байт | 0-4 байт | 0-4 байта |

* REX – префикс, позволяющий расширить число адресуемых регистров для поддержки 64-битного адресного пространства.
* Opcode – аналогично MIPS показывает тип инструкции.
* Mod\_R/M – используется для адресации операндов. Может отсутствовать в кодировке, если инструкция не имеет явных операндов.
* SIB (Scale index base) – второй байт, использующийся для адресации операндов в памяти. Может отсутствовать.
* Displacement – байт смещения.
* Immediate - константа

## Язык ассемблера x86.

Язык ассемблера x86 – это семейство обратно совместимых языков ассемблера, которые обеспечивают определенный уровень совместимости.

На данный момент язык поддерживает более 1000 инструкций, работу с целыми и вещественными числами, а также работу с векторами и прерываниями.

Подробный обзор инструкций можно увидеть на сайте Википедии: https://ru.qaz.wiki/wiki/X86\_instruction\_listings