

Процессор МУР128
(машина учебная регистровая, 128–разрядная)

Гаврилов В.С.

16 декабря 2017 г.

Оглавление

1	Основы архитектуры	3
1.1	Регистры	3
1.2	Режимы адресации	3
2	Список команд	4
2.1	Целочисленные команды	4
2.1.1	Команды пересылки	4
2.1.2	Арифметические команды	5
2.1.3	Поразрядные операции	6
2.2	Вещественные команды	7
2.2.1	Команды пересылки	7
2.2.2	Арифметические команды	7
2.3	Команды передачи управления	8
3	Кодировка команд	11

Список таблиц

2.1.1 Целочисленные команды пересылки	4
2.1.1 Целочисленные команды пересылки	5
2.1.2 Целочисленные арифметические команды	5
2.1.2 Целочисленные арифметические команды	6
2.1.3 Команды поразрядных операций	6
2.2.1 Вещественные команды пересылки	7
2.2.2 Вещественные арифметические команды	7
2.2.2 Вещественные арифметические команды	8
2.3.1 Команды передачи управления	8
2.3.1 Команды передачи управления	9
2.3.1 Команды передачи управления	10
2.3.2 Обозначения условий для условной передачи управления	10

Глава 1. Основы архитектуры

1.1. Регистры

Процессор MYP128 имеет следующие регистры: 32 целочисленных регистра общего назначения (r0–r31) по 128 разрядов в каждом; 32 вещественных регистра (f0–f31), тоже по 128 разрядов в каждом; и 128-разрядный указатель команд ip. Регистры общего назначения можно использовать для целых чисел и для указателей. Вещественные регистры предназначены для хранения вещественных чисел. Формат хранения данных в вещественных регистрах — формат четырёхкратной точности. Имеется также указатель стека sp и указатель кадра стека, bp. Первый из них идентичен r31, а второй идентичен r30. Специального регистра флагов нет.

1.2. Режимы адресации

Находящийся в памяти операнд можно адресовать с помощью одной из следующих форм:

$$\text{Address} = \text{BP} + \text{IX} * \text{SF}$$

$$\text{Address} = \text{BP} + \text{OS}$$

Здесь BP — 128-разрядный базовый указатель, IX — 128-разрядный индексный регистр, SF — масштабирующий множитель, а OS — непосредственно заданное смещение. Базовый указатель присутствует всегда, остальные элементы — необязательны.

Базовый указатель, BP, может быть регистром общего назначения, в том числе указателем стека (sp) и указателем кадра стека (bp).

Индексный регистр, IX, может быть одним из регистров r0–r30. Значение, равное 31, означает отсутствие индексного регистра.

Масштабирующий множитель, SF, может принимать значения 1, 2, 4, 8, 10, 16.

Смещение, OS, представляет собой расширенное знаком десятиразрядное число.

Переходы и вызовы могут использовать как смещение относительно указателя команд, так и абсолютный адрес. В каждом из этих случаев адрес перехода может указываться как в регистре общего назначения, так и непосредственно заданной 15-разрядной константой, расширяемой знаком до 128 разрядов. При использовании в качестве адреса перехода смещения относительно указателя команд, адрес перехода вычисляется так:

$$\text{Address} = \text{IP} + 4 * \text{Offset}$$

поскольку размер любой команды процессора MYP128 равен 4 байтам.

Глава 2. Список команд

2.1. Целочисленные команды

Целочисленные команды можно разделить на три больших группы: команды пересылки, арифметические команды, и поразрядные операции. В следующих трёх разделах перечисляются все эти команды и описывается их поведение.

2.1.1. Команды пересылки

К целочисленным командам пересылки относятся следующие команды:

Таблица 2.1.1. Целочисленные команды пересылки

Команда	Пояснение
mov reg1, reg2	Записать в целочисленный регистр reg1 содержимое целочисленного регистра reg2.
movu reg, imm15	Записать в целочисленный регистр reg непосредственно заданную константу imm15, расширив её нулём до 128 разрядов.
movs reg, imm15	Записать в целочисленный регистр reg непосредственно заданную константу imm15, расширив её знаком до 128 разрядов.
mov reg, [mem128]	Загрузить в целочисленный регистр reg содержимое 128-разрядной ячейки памяти mem128.
mov8u reg, [mem8]	Загрузить в целочисленный регистр reg содержимое 8-разрядной ячейки памяти mem8, расширив его перед этим нулём до 128 разрядов.
mov16u reg, [mem16]	Загрузить в целочисленный регистр reg содержимое 16-разрядной ячейки памяти mem16, расширив его перед этим нулём до 128 разрядов.
mov32u reg, [mem32]	Загрузить в целочисленный регистр reg содержимое 32-разрядной ячейки памяти mem32, расширив его перед этим нулём до 128 разрядов.
mov64u reg, [mem64]	Загрузить в целочисленный регистр reg содержимое 64-разрядной ячейки памяти mem64, расширив его перед этим нулём до 128 разрядов.
mov8s reg, [mem8]	Загрузить в целочисленный регистр reg содержимое 8-разрядной ячейки памяти mem8, расширив его перед этим знаком до 128 разрядов.
mov16s reg, [mem16]	Загрузить в целочисленный регистр reg содержимое 16-разрядной ячейки памяти mem16, расширив его перед этим знаком до 128 разрядов.
mov32s reg, [mem32]	Загрузить в целочисленный регистр reg содержимое 32-разрядной ячейки памяти mem32, расширив его перед этим знаком до 128 разрядов.
mov64s reg, [mem64]	Загрузить в целочисленный регистр reg содержимое 64-разрядной ячейки памяти mem64, расширив его перед этим знаком до 128 разрядов.
mov [mem128], reg	Сохранить содержимое целочисленного регистра reg в 128-разрядной ячейке памяти mem128.
mov8 [mem8], reg	Сохранить младшие 8 разрядов целочисленного регистра reg в 8-разрядной ячейке памяти mem8.
mov16 [mem16], reg	Сохранить младшие 16 разрядов целочисленного регистра reg в 16-разрядной ячейке памяти mem16.
mov32 [mem32], reg	Сохранить младшие 32 разряда целочисленного регистра reg в 32-разрядной ячейке памяти mem32.
mov64 [mem64], reg	Сохранить младшие 64 разряда целочисленного регистра reg в 64-разрядной ячейке памяти mem64.

Таблица 2.1.1. Целочисленные команды пересылки

Команда	Пояснение
push $ri-rj$	Сохранить в стеке содержимое регистров с ri по rj (должно быть $i \leq j$). А именно, делается следующее: <pre>for(n = i; n <= j; n++){ sp -= 16; *sp ← rn; }</pre>
pop $ri-rj$	Восстановить из стека содержимое регистров с ri по rj (должно быть $i \leq j$). А именно, делается следующее: <pre>for(n = i; n <= j; n++){ rn ← *sp; sp += 16; }</pre>

2.1.2. Арифметические команды

К арифметическим командам относятся команды целочисленных сложения, вычитания, умножения, деления, и вычисления остатка, а также команды сравнения.

В следующей таблице перечислены все целочисленные арифметические команды.

Таблица 2.1.2. Целочисленные арифметические команды

Команда	Пояснение
addi reg1, reg2, reg3	$reg1 \leftarrow reg2 + reg3$
addi reg1, reg2, imm10	$reg1 \leftarrow reg2 + imm10$
subi reg1, reg2, reg3	$reg1 \leftarrow reg2 - reg3$
subi reg1, reg2, imm10	$reg1 \leftarrow reg2 - imm10$
muliu reg1, reg2, reg3	Записать в reg1 младшие 128 разрядов результата беззнакового произведения reg2 и reg3.
muliu reg1, reg2, imm10	Записать в reg1 младшие 128 разрядов результата беззнакового произведения reg2 и imm10.
mulis reg1, reg2, reg3	Записать в reg1 младшие 128 разрядов результата знакового произведения reg2 и reg3.
mulis reg1, reg2, imm10	Записать в reg1 младшие 128 разрядов результата знакового произведения reg2 и imm10.
diviu reg1, reg2, reg3	Записать в reg1 частное при беззнаковом делении reg2 на reg3.
diviu reg1, reg2, imm10	Записать в reg1 частное при беззнаковом делении reg2 на imm10.
divis reg1, reg2, reg3	Записать в reg1 частное при знаковом делении reg2 на reg3.
divis reg1, reg2, imm10	Записать в reg1 частное при беззнаковом делении reg2 на imm10.
modiu reg1, reg2, reg3	Записать в reg1 остаток от беззнакового деления reg2 на reg3.
modiu reg1, reg2, imm10	Записать в reg1 остаток от беззнакового деления reg2 на imm10.
modis reg1, reg2, reg3	Записать в reg1 остаток от знакового деления reg2 на reg3.
modis reg1, reg2, imm10	Записать в reg1 остаток от знакового деления reg2 на imm10.
divmodiu reg1, reg2, reg3, reg4	Записать в reg1 частное при беззнаковом делении reg3 на reg4, а в reg2 — остаток от деления.
divmodis reg1, reg2, reg3, reg4	Записать в reg1 частное при знаковом делении reg3 на reg4, а в reg2 — остаток от деления.
cmpiu reg1, reg2, reg3	Беззнаково сравнить reg2 и reg3. Далее так: $reg1 \leftarrow \begin{cases} -1, & \text{если } reg2 < reg3; \\ 0, & \text{если } reg2 = reg3; \\ +1, & \text{если } reg2 > reg3. \end{cases}$

Таблица 2.1.2. Целочисленные арифметические команды

Команда	Пояснение
cmpis reg1, reg2, reg3	Знаково сравнить reg2 и reg3. Далее так: $\text{reg1} \leftarrow \begin{cases} -1, & \text{если } \text{reg2} < \text{reg3}; \\ 0, & \text{если } \text{reg2} = \text{reg3}; \\ +1, & \text{если } \text{reg2} > \text{reg3}. \end{cases}$
cmpiu reg1, reg2, imm10	Беззнаково сравнить reg2 и imm10. Далее так: $\text{reg1} \leftarrow \begin{cases} -1, & \text{если } \text{reg2} < \text{imm10}; \\ 0, & \text{если } \text{reg2} = \text{imm10}; \\ +1, & \text{если } \text{reg2} > \text{imm10}. \end{cases}$
cmpis reg1, reg2, imm10	Знаково сравнить reg2 и imm10. Далее так: $\text{reg1} \leftarrow \begin{cases} -1, & \text{если } \text{reg2} < \text{imm10}; \\ 0, & \text{если } \text{reg2} = \text{imm10}; \\ +1, & \text{если } \text{reg2} > \text{imm10}. \end{cases}$

В данном разделе и в последующих imm10 — 10-разрядная знаковая константа.

2.1.3. Поразрядные операции

В приводимой ниже таблице сведены все команды вычисления поразрядных операций:

Таблица 2.1.3. Команды поразрядных операций

Команда	Пояснение
and reg1, reg2, reg3	$\text{reg1} \leftarrow \text{reg2} \& \text{reg3}$
and reg1, reg2, imm10	$\text{reg1} \leftarrow \text{reg2} \& \text{imm10}$
or reg1, reg2, reg3	$\text{reg1} \leftarrow \text{reg2} \text{reg3}$
or reg1, reg2, imm10	$\text{reg1} \leftarrow \text{reg2} \text{imm10}$
xor reg1, reg2, reg3	$\text{reg1} \leftarrow \text{reg2} \wedge \text{reg3}$
xor reg1, reg2, imm10	$\text{reg1} \leftarrow \text{reg2} \wedge \text{imm10}$
andn reg1, reg2, reg3	$\text{reg1} \leftarrow \text{reg2} \& \sim \text{reg3}$
andn reg1, reg2, imm10	$\text{reg1} \leftarrow \text{reg2} \& \sim \text{imm10}$
orn reg1, reg2, reg3	$\text{reg1} \leftarrow \text{reg2} \sim \text{reg3}$
orn reg1, reg2, imm10	$\text{reg1} \leftarrow \text{reg2} \sim \text{imm10}$
xorn reg1, reg2, reg3	$\text{reg1} \leftarrow \text{reg2} \wedge \sim \text{reg3}$
xorn reg1, reg2, imm10	$\text{reg1} \leftarrow \text{reg2} \wedge \sim \text{imm10}$
lshift reg1, reg2, reg3	$\text{reg1} \leftarrow \text{reg2} \ll \text{reg3}$
lshift reg1, reg2, imm10	$\text{reg1} \leftarrow \text{reg2} \ll \sim \text{imm10}$
rshift reg1, reg2, reg3	$\text{reg1} \leftarrow \text{reg2} \gg \text{reg3}$ (сдвиг вправо беззнакового числа, при этом слева вдвигаются нули)
rshift reg1, reg2, imm10	$\text{reg1} \leftarrow \text{reg2} \gg \sim \text{imm10}$ (сдвиг вправо беззнакового числа, при этом слева вдвигаются нули)
rshifts reg1, reg2, reg3	$\text{reg1} \leftarrow \text{reg2} \gg \text{reg3}$ (сдвиг вправо знакового числа, при этом слева остаётся неизменным знаковый разряд)
rshifts reg1, reg2, imm10	$\text{reg1} \leftarrow \text{reg2} \gg \text{imm10}$ (сдвиг вправо знакового числа, при этом слева остаётся неизменным знаковый разряд)
not reg1, reg2	$\text{reg1} \leftarrow \sim \text{reg2}$
not reg1, imm15	$\text{reg1} \leftarrow \sim \text{imm15}$

В данном разделе и в последующих imm15 — 15-разрядная знаковая константа.

2.2. Вещественные команды

Вещественные команды можно разделить на две больших группы: команды пересылки и арифметические команды. В следующих двух разделах перечисляются все эти команды и описывается их поведение.

2.2.1. Команды пересылки

К целочисленным командам пересылки относятся следующие команды:

Таблица 2.2.1. Вещественные команды пересылки

Команда	Пояснение
mov reg1, reg2	Записать в вещественный регистр reg1 содержимое вещественного регистра reg2.
mov reg, [mem128]	Загрузить в вещественный регистр reg содержимое 128-разрядной ячейки памяти mem128.
mov32f reg, [mem32]	Загрузить в вещественный регистр reg содержимое 32-разрядной ячейки памяти mem32.
mov64f reg, [mem64]	Загрузить в вещественный регистр reg содержимое 64-разрядной ячейки памяти mem64.
mov80f reg, [mem80]	Загрузить в вещественный регистр reg содержимое 80-разрядной ячейки памяти mem80.
mov32f [mem32], reg	Сохранить содержимое вещественного регистра reg в 32-разрядной ячейке памяти mem32 (происходит преобразование из четырёхкратной точности к одинарной).
mov64f [mem64], reg	Сохранить содержимое вещественного регистра reg в 64-разрядной ячейке памяти mem64 (происходит преобразование из четырёхкратной точности к двойной точности).
mov80f [mem64], reg	Сохранить содержимое вещественного регистра reg в 80-разрядной ячейке памяти mem80 (происходит преобразование из четырёхкратной точности к расширенной точности).
push fi-fj	Сохранить в стеке содержимое регистров с f_i по f_j (должно быть $i \leq j$). А именно, делается следующее: <pre>for(n = i; n <= j; n++){ sp -= 16; *sp ← fn; }</pre>
pop fi-fj	Восстановить из стека содержимое регистров с f_i по f_j (должно быть $i \leq j$). А именно, делается следующее: <pre>for(n = i; n <= j; n++){ fn ← *sp; sp += 16; }</pre>

2.2.2. Арифметические команды

К вещественным арифметическим командам относятся команды выполнения четырёх действий арифметики, команда сравнения, и команды округления. Все вещественные арифметические команды собраны в приводимой ниже таблице.

Таблица 2.2.2. Вещественные арифметические команды

Команда	Пояснение
addf reg1, reg2, reg3	$reg1 \leftarrow reg2 + reg3$
subf reg1, reg2, reg3	$reg1 \leftarrow reg2 - reg3$
mulf reg1, reg2, reg3	$reg1 \leftarrow reg2 * reg3$
divf reg1, reg2, reg3	$reg1 \leftarrow reg2 / reg3$

Таблица 2.2.2. Вещественные арифметические команды

Команда	Пояснение
cmpf reg1, reg2, reg3	Сравнить содержимое вещественных регистров reg2 и reg3, а результат записать в целочисленный регистр reg1. А именно: $\text{reg1} \leftarrow \begin{cases} -1, & \text{если } \text{reg2} < \text{reg3}; \\ 0, & \text{если } \text{reg2} = \text{reg3}; \\ +1, & \text{если } \text{reg2} > \text{reg3}; \\ +2, & \text{если содержимое reg2 и reg3 — несравнимо.} \end{cases}$
roundn reg1, reg2	Записать в целочисленный регистр reg1 результат округления содержимого вещественного регистра reg2 к ближайшему целому.
roundl reg1, reg2	Записать в целочисленный регистр reg1 результат округления содержимого вещественного регистра reg2 к ближайшему меньшему целому.
roundg reg1, reg2	Записать в целочисленный регистр reg1 результат округления содержимого вещественного регистра reg2 к ближайшему большему целому.
roundt reg1, reg2	Записать в целочисленный регистр reg1 результат округления содержимого вещественного регистра reg2 отбрасыванием дробной части.
frac reg1, reg2	Записать в вещественный регистр reg1 дробную часть содержимого вещественного регистра reg2.

2.3. Команды передачи управления

К командам передачи управления относятся команды условного и безусловного вызова подпрограммы, условного и безусловного перехода, условного и безусловного возврата, команда возбуждения программного прерывания, команда возврата из прерывания. В командах условного и безусловного перехода, а также в командах условного и безусловного вызова подпрограммы адрес перехода может быть как абсолютным, так и смещением относительно регистра указателя команд ip.

В следующей таблице приведены все команды передачи управления.

Таблица 2.3.1. Команды передачи управления

Команда	Пояснение
jmp reg	Безусловный переход по абсолютному адресу, указанному в регистре: $\text{ip} \leftarrow \text{reg}$, где reg — целочисленный регистр
jmp r reg	Безусловный относительный переход: $\text{ip} += 4 * \text{reg}$, где reg — целочисленный регистр
jmp r imm20	Безусловный относительный переход: $\text{ip} += 4 * \text{imm20}$, где imm20 — 20-разрядная знаковая константа.
jmpcc reg1, reg2	Условный переход по абсолютному адресу, указанному в регистре: если содержимое целочисленного регистра reg1 удовлетворяет условию cc, то $\text{ip} \leftarrow \text{reg2}$, где reg2 — целочисленный регистр
jmpcc r reg1, reg2	Условный относительный переход: если содержимое целочисленного регистра reg1 удовлетворяет условию cc, то $\text{ip} += 4 * \text{reg2}$, где reg2 — целочисленный регистр
jmpcc r reg, imm15	Условный относительный переход: если содержимое целочисленного регистра reg удовлетворяет условию cc, то $\text{ip} += 4 * \text{imm20}$, где imm15 — 15-разрядная знаковая константа.
call reg	Безусловный вызов подпрограммы, абсолютный адрес которой находится в целочисленном регистре reg. А именно, делается следующее: $\text{sp} -= 16;$ $*\text{sp} = \text{ip};$ $\text{ip} \leftarrow \text{reg};$

Таблица 2.3.1. Команды передачи управления

Команда	Пояснение
callr reg	Безусловный вызов подпрограммы, относительный адрес которой находится в целочисленном регистре reg. А именно, делается следующее: <pre> sp -= 16; *sp = ip; ip += 4 * reg; </pre>
callr imm20	Безусловный вызов подпрограммы, относительный адрес которой задан знаковой константой imm20. А именно, делается следующее: <pre> sp -= 16; *sp = ip; ip += 4 * imm20; </pre>
callcc reg1, reg2	Условный вызов подпрограммы: если содержимое целочисленного регистра reg1 удовлетворяет условию cc, то перейти к выполнению подпрограммы, абсолютный адрес которой указан в целочисленном регистре reg2. А именно, делается следующее: <pre> if(cc){ sp -= 16; *sp = ip; ip ← reg2; } </pre>
callccr reg1, reg2	Условный вызов подпрограммы: если содержимое целочисленного регистра reg1 удовлетворяет условию cc, то перейти к выполнению подпрограммы, относительный адрес которой указан в целочисленном регистре reg2. А именно, делается следующее: <pre> if(cc){ sp -= 16; *sp = ip; ip ← 4 * reg2; } </pre>
callccr reg, imm15	Условный вызов подпрограммы: если содержимое целочисленного регистра reg удовлетворяет условию cc, то перейти к выполнению подпрограммы, относительный адрес которой указан непосредственно заданной 15-разрядной знаковой константой imm15. А именно, делается следующее: <pre> if(cc){ sp -= 16; *sp = ip; ip ← 4 * imm15; } </pre>
ret	Безусловный возврат из подпрограммы. А именно, делается следующее: <pre> ip ← *sp; sp += 16; </pre>
retcc reg	Условный возврат из подпрограммы, т.е. возврат из подпрограммы, если содержимое целочисленного регистра reg удовлетворяет условию cc. А именно, делается следующее: <pre> if(cc){ ip ← *sp; sp += 16; } </pre>
reta reg	Безусловный возврат из подпрограммы с очисткой стека от переданных аргументов. В целочисленном регистре reg содержится размер переданных аргументов в байтах. А именно, делается следующее: <pre> ip ← *sp; sp += 16; sp += reg; </pre>

Таблица 2.3.1. Команды передачи управления

Команда	Пояснение
reta imm20	Безусловный возврат из подпрограммы с очисткой стека от переданных аргументов. Непосредственно заданная беззнаковая константа imm20 равна размеру переданных аргументов в байтах. А именно, делается следующее: $ip \leftarrow *sp;$ $sp += 16;$ $sp += imm20;$
retacc reg1, reg2	Условный возврат из подпрограммы с очисткой стека от переданных аргументов, если содержимое целочисленного регистра reg1 удовлетворяет условию cc. В целочисленном регистре reg2 содержится размер переданных аргументов в байтах. А именно, делается следующее: if(cc){ $ip \leftarrow *sp;$ $sp += 16;$ $sp += reg2;$ }
retacc reg, imm15	Условный возврат из подпрограммы с очисткой стека от переданных аргументов, если содержимое целочисленного регистра reg удовлетворяет условию cc. Беззнаковая целочисленная константа imm15 определяет размер переданных аргументов в байтах. А именно, делается следующее: if(cc){ $ip \leftarrow *sp;$ $sp += 16;$ $sp += imm15;$ }
trap reg	Вызов подпрограммы обработки прерывания, номер которого задан в целочисленном регистре reg. А именно, делается следующее: push r0-r31; push ip; $ip \leftarrow (reg \& 0x3FF) * 16;$
trap imm10	Вызов подпрограммы обработки прерывания, номер которого задан беззнаковой целочисленной константой imm10. А именно, делается следующее: push r0-r31; push ip; $ip \leftarrow imm10 * 16;$
reti	Возврат из подпрограммы обработки прерывания. А именно, делается следующее: pop ip; pop r0-r31;

Здесь cc — условие, которому должно удовлетворять содержимое регистра, чтобы произошла передача управления. В приводимой ниже таблице приведены все такие условия.

Таблица 2.3.2. Обозначения условий для условной передачи управления

Обозначение	Пояснение
s	Содержимое проверяемого регистра — отрицательно.
z	Содержимое проверяемого регистра — ноль.
p	Содержимое проверяемого регистра — положительно.
nz	Содержимое проверяемого регистра — не ноль.
ge	Содержимое проверяемого регистра — больше или равно нулю.
le	Содержимое проверяемого регистра — меньше или равно нулю.

Глава 3. Кодировка команд