Процессор МУР128
(машина учебная регистровая, 128–разрядная)

Гаврилов В.С.

17 декабря 2017 г.

Оглавление

Список таблиц

Глава 1. Основы архитектуры

1.1. Регистры

Процессор МУР128 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 следующей формы:

Address = BP + IX*SF

Здесь BP — 128–разрядный базовый указатель, IX — 128–разрядный индексный регистр, SF — масштабирующий множитель.

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.

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

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:

Address = IP + 4 * Offset

поскольку размер любой команды процессора МУР128 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 $i \leqslant j$). А именно, делается следующее:
	`for(n = i; n <= j; n++){`
	`  sp -= 16;`
	`  *sp ← rn;`
	`}`

pop ri-rj	Восстановить из стека содержимое регистров с ri по rj (должно быть $i \leqslant j$). А именно, делается следующее:
	`for(n = i; n <= j; n++){`
	`  rn ← *sp;`
	`  sp += 16;`
	`}`

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: $\texttt{reg1} \leftarrow \begin{cases} -1, \text{ если } \texttt{reg2} < \texttt{reg3}; \\ 0, \text{ если } \texttt{reg2} = \texttt{reg3}; \\ +1, \text{ если } \texttt{reg2} > \texttt{reg3}. \end{cases}$

Таблица 2.1.2. Целочисленные арифметические команды

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

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

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 fi по fj (должно быть $i \leqslant j$). А именно, делается следующее:
	`for(n = i; n <= j; n++){`
	`  sp -= 16;`
	`  *sp` $\leftarrow$ fn`;`
	`}`

pop fi-fj	Восстановить из стека содержимое регистров с fi по fj (должно быть $i \leqslant j$). А именно, делается следующее:
	`for(n = i; n <= j; n++){`
	`  f`n $\leftarrow$ `*sp;`
	`  sp += 16;`
	`}`

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: $\texttt{reg1} \leftarrow \begin{cases} -1, \text{ если } \texttt{reg2} < \texttt{reg3}; \\ 0, \text{ если } \texttt{reg2} = \texttt{reg3}; \\ +1, \text{ если } \texttt{reg2} > \texttt{reg3}; \\ +2, \text{ если содержимое reg2 и reg3 — несравнимо.} \end{cases}$
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: ip ← reg, где reg — целочисленный регистр
jmpr reg	Безусловный относительный переход: ip += 4 * reg, где reg — целочисленный регистр
jmpr imm20	Безусловный относительный переход: ip += 4 * imm20, где imm20 — 20–разрядная знаковая константа.
jmp*cc* reg1, reg2	Условный переход по абсолютному адресу, указанному в регистре: если содержимое целочисленного регистра reg1 удовлетворяет условию *cc*, то ip ← reg2, где reg2 — целочисленный регистр
jmp*cc*r reg1, reg2	Условный относительный переход: если содержимое целочисленного регистра reg1 удовлетворяет условию *cc*, то ip += 4 * reg2, где reg2 — целочисленный регистр
jmp*cc*r reg, imm15	Условный относительный переход: если содержимое целочисленного регистра reg удовлетворяет условию *cc*, то ip += 4 * imm20, где imm15 — 15–разрядная знаковая константа.

call reg	Безусловный вызов подпрограммы, абсолютный адрес которой находится в целочисленном регистре reg. А именно, делается следующее:
	`sp -= 16;`
	`*sp = ip;`
	`ip ← 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:
	`sp -= 16;`
	`*sp = ip;`
	`ip += 4 * reg;`

callr imm20	Безусловный вызов подпрограммы, относительный адрес которой задан знаковой константой imm20. А именно, делается следующее:
	`sp -= 16;`
	`*sp = ip;`
	`ip += 4 * imm20;`

call*cc* 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:
	`if(cc){`
	`sp -= 16;`
	`*sp = ip;`
	`ip ← reg2;`
	`}`

call*cc*r 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:
	`if(cc){`
	`sp -= 16;`
	`*sp = ip;`
	`ip ← 4 * reg2;`
	`}`

call*cc*r 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:
	`if(cc){`
	`sp -= 16;`
	`*sp = ip;`
	`ip ← 4 * imm15;`
	`}`

ret	Безусловный возврат из подпрограммы. А именно, делается следующее:
	`ip ← *sp;`
	`sp += 16;`

ret*cc* reg	Условный возврат из подпрограммы, т.е. возврат из подпрограммы, если содержимое целочисленного регистра reg удовлетворяет условию *cc*. А именно, делается следующее:
	`if(cc){`
	`ip ← *sp;`
	`sp += 16;`
	`}`

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:
	`ip ← *sp;`
	`sp += 16;`
	`sp += reg;`

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 ← *sp;`
	`sp += 16;`
	`sp += imm20;`

reta*cc* 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 ← *sp;`
	`sp += 16;`
	`sp += reg2;`
	`}`

reta*cc* 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 ← *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 ← (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 ← 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 нуля.
le	Содержимое проверяемого регистра — меньше или равно нуля.

Глава 3. Кодировка команд

В настоящей главе описываются форматы команд и кодировка команд.

3.1. Формат команд

Все команды используют один из общих шаблонов форматов, показанных ниже (самые старшие разряды — слева).

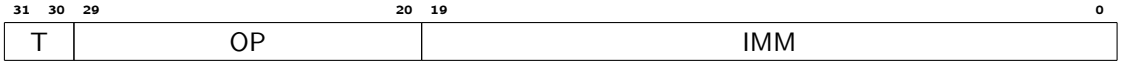

Рис. 3.1.1. Шаблон A. Имеется один операнд, являющийся непосредственно заданной константой.

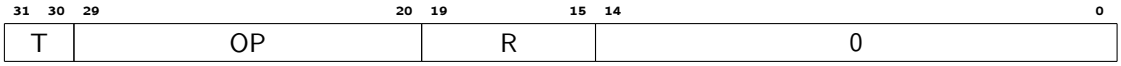

Рис. 3.1.2. Шаблон B. Имеется один регистровый операнд.

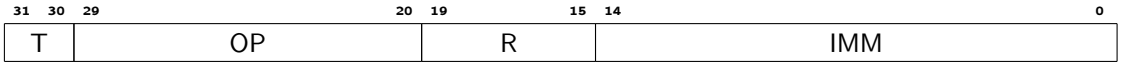

Рис. 3.1.3. Шаблон C. Имеется один регистровый операнд и один операнд, являющийся непосредственно заданной константой.

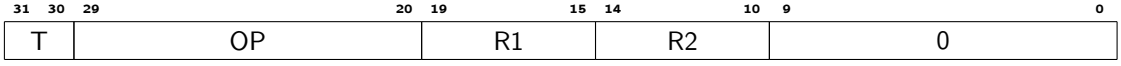

Рис. 3.1.4. Шаблон D. Имеется два регистровых операнда.

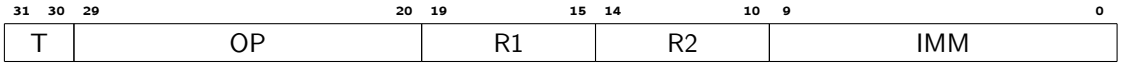

Рис. 3.1.5. Шаблон E. Имеется два регистровых операнда и один операнд, являющийся непосредственно заданной константой.

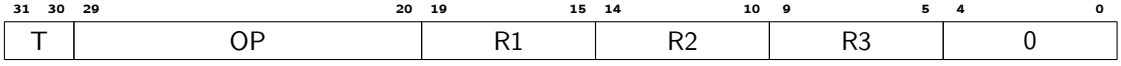

Рис. 3.1.6. Шаблон F. Имеется три регистровых операнда.

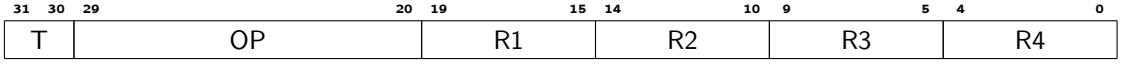

Рис. 3.1.7. Шаблон G. Имеется четыре регистровых операнда.

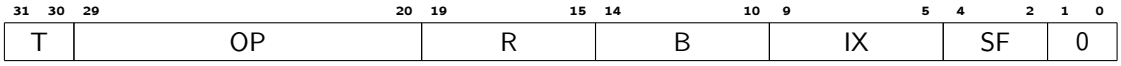

Рис. 3.1.8. Шаблон M. Команда загрузки или сохранения.

В следующих таблица приведены коды регистров, коды условий, и коды в поле SF.

Таблица 3.1.1. Коды целочисленных регистров

Регистр	**Код**
r0	00000
r1	00001
r2	00010
r3	00011
r4	00100
r5	00101
r6	00110
r7	00111
r8	01000
r9	01001
r10	01010
r11	01011
r12	01100
r13	01101
r14	01110
r15	01111
r16	10000
r17	10001
r18	10010
r19	10011
r20	10100
r21	10101
r22	10110
r23	10111
r24	11000
r25	11001
r26	11010
r27	11011
r28	11100
r29	11101
r30	11110
r31	11111

Таблица 3.1.2. Коды вещественных регистров

Регистр	**Код**
f0	00000
f1	00001
f2	00010
f3	00011
f4	00100
f5	00101
f6	00110
f7	00111
f8	01000
f9	01001
f10	01010
f11	01011
f12	01100
f13	01101
f14	01110
f15	01111
f16	10000

Таблица 3.1.2. Коды вещественных регистров

Регистр	Код
f17	10001
f18	10010
f19	10011
f20	10100
f21	10101
f22	10110
f23	10111
f24	11000
f25	11001
f26	11010
f27	11011
f28	11100
f29	11101
f30	11110
f31	11111

Таблица 3.1.3. Коды условий

Условие	Код
s	000
z	001
p	010
nz	011
ge	100
le	101

Таблица 3.1.4. Коды в поле SF

Масштабирующий множитель	Код
1	000
2	001
4	010
8	011
16	100
10	101

Коды 110 и 111 в поле SF — зарезервированы.
Поле T отвечает за тип команды, поле OP — за код операции. Возможны следующие значения поля T:

Таблица 3.1.5. Коды в поле T

Код	Пояснения
00	целочисленная команда, но не команда пересылки
01	вещественная команда, но не команда пересылки
10	команда передачи управления
11	команда пересылки

В последующих разделах подробно описана кодирование каждой команды.

3.2. Целочисленные команды

3.2.1. Команды пересылки

В следующей таблице описано кодирование целочисленных команд пересылки. Двоеточием разделяются поля машинной команды. В двоичном коде команды под reg, reg1, reg2, reg3, reg4, ri, rj понимаются коды

соответствующих регистровых аргументов команды, взятые из таблицы 3.1.1. Под imm10, imm15, imm20 в двоичном коде понимается двоичное представление соответствующей константы, являющейся аргументом команды.

Таблица 3.2.1. Кодирование целочисленных команд пересылки

Команда	**Двоичный код**
mov reg1, reg2	11:0000000000:reg1:reg2:0000000000 (т.е. используется шаблон D с T $= 11$, OP $=$ 0000000000, R1 $=$ reg1, R2 $=$ reg2)
movu reg, imm15	11:0000000001:reg:imm15 (т.е. используется шаблон C с T $= 11$, OP $= 0000000001$, R $=$ reg)
movs reg, imm15	11:0000000010:reg:imm15 (т.е. используется шаблон C с T $= 11$, OP $= 0000000010$, R $=$ reg)
mov reg, [mem128]	11:0000000011:reg:B:IX:SF:00 (т.е. используется шаблон M с T $= 11$, OP $=$ 0000000011, R $=$ reg)
mov8u reg, [mem8]	11:0000000100:reg:B:IX:SF:00 (т.е. используется шаблон M с T $= 11$, OP $=$ 0000000100, R $=$ reg)
mov16u reg, [mem16]	11:0000000101:reg:B:IX:SF:00 (т.е. используется шаблон M с T $= 11$, OP $=$ 0000000101, R $=$ reg)
mov32u reg, [mem32]	11:0000000110:reg:B:IX:SF:00 (т.е. используется шаблон M с T $= 11$, OP $=$ 0000000110, R $=$ reg)
mov64u reg, [mem32]	11:0000000111:reg:B:IX:SF:00 (т.е. используется шаблон M с T $= 11$, OP $=$ 0000000111, R $=$ reg)
mov8s reg, [mem8]	11:0000001000:reg:B:IX:SF:00 (т.е. используется шаблон M с T $= 11$, OP $=$ 0000001000, R $=$ reg)
mov16u reg, [mem16]	11:0000001001:reg:B:IX:SF:00 (т.е. используется шаблон M с T $= 11$, OP $=$ 0000001001, R $=$ reg)
mov32u reg, [mem32]	11:0000001010:reg:B:IX:SF:00 (т.е. используется шаблон M с T $= 11$, OP $=$ 0000001010, R $=$ reg)
mov64u reg, [mem32]	11:0000001011:reg:B:IX:SF:00 (т.е. используется шаблон M с T $= 11$, OP $=$ 0000001011, R $=$ reg)
mov [mem128], reg	11:0000001100:reg:B:IX:SF:00 (т.е. используется шаблон M с T $= 11$, OP $=$ 0000001100, R $=$ reg)
mov8 [mem8], reg	11:0000001101:reg:B:IX:SF:00 (т.е. используется шаблон M с T $= 11$, OP $=$ 0000001101, R $=$ reg)
mov16 [mem16], reg	11:0000001110:reg:B:IX:SF:00 (т.е. используется шаблон M с T $= 11$, OP $=$ 0000001110, R $=$ reg)
mov32 [mem32], reg	11:0000001111:reg:B:IX:SF:00 (т.е. используется шаблон M с T $= 11$, OP $=$ 0000001111, R $=$ reg)
mov64 [mem64], reg	11:0000010000:reg:B:IX:SF:00 (т.е. используется шаблон M с T $= 11$, OP $=$ 0000010000, R $=$ reg)
push ri-rj	11:0000010001:ri:rj:0000000000 (т.е. используется шаблон D с T $= 11$, OP $=$ 0000010001, R1 $=$ ri, R2 $=$ rj)
pop ri-rj	11:0000010010:ri:rj:0000000000 (т.е. используется шаблон D с T $= 11$, OP $=$ 0000010010, R1 $=$ ri, R2 $=$ rj)