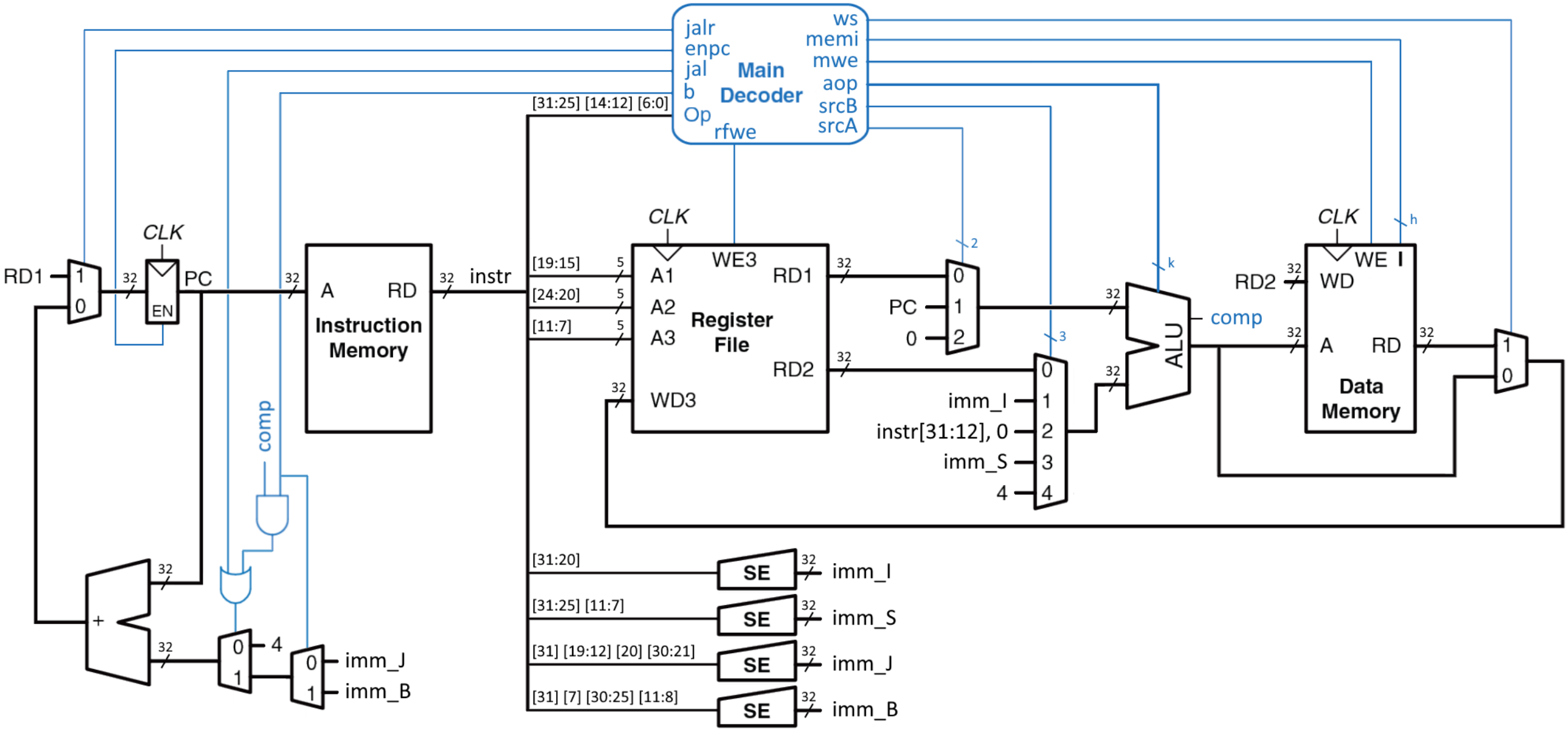
4

Тракт данных

Архитектуры микропроцессорных систем и средств


jalr
enpc
jal
b
Op
Main Decoder
rfwe
ws
memi
mwe
aop
srcB
srcA
[31:25] [14:12] [6:0]
CLK
RD1
PC
EN
A
RD
Instruction Memory
instr
[19:15]
[24:20]
[11:7]
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
PC
imm_I
instr[31:12], 0
imm_S
4
ALU
comp
RD2
WD
WE
Data Memory
+
imm_J
imm_B
[31:20]
[31:25] [11:7]
[31] [19:12] [20] [30:21]
[31] [7] [30:25] [11:8]
SE

Система команд RISC-V

Базовые наборы команд:

RV32I	содержит инструкции, оперирующие 32-битными целыми числами
RV64I	содержит инструкции, оперирующие 64-битными целыми числами
RV128I	содержит инструкции, оперирующие 128-битными целыми числами
RV32E	сокращенная версия RV32I для встраиваемых систем

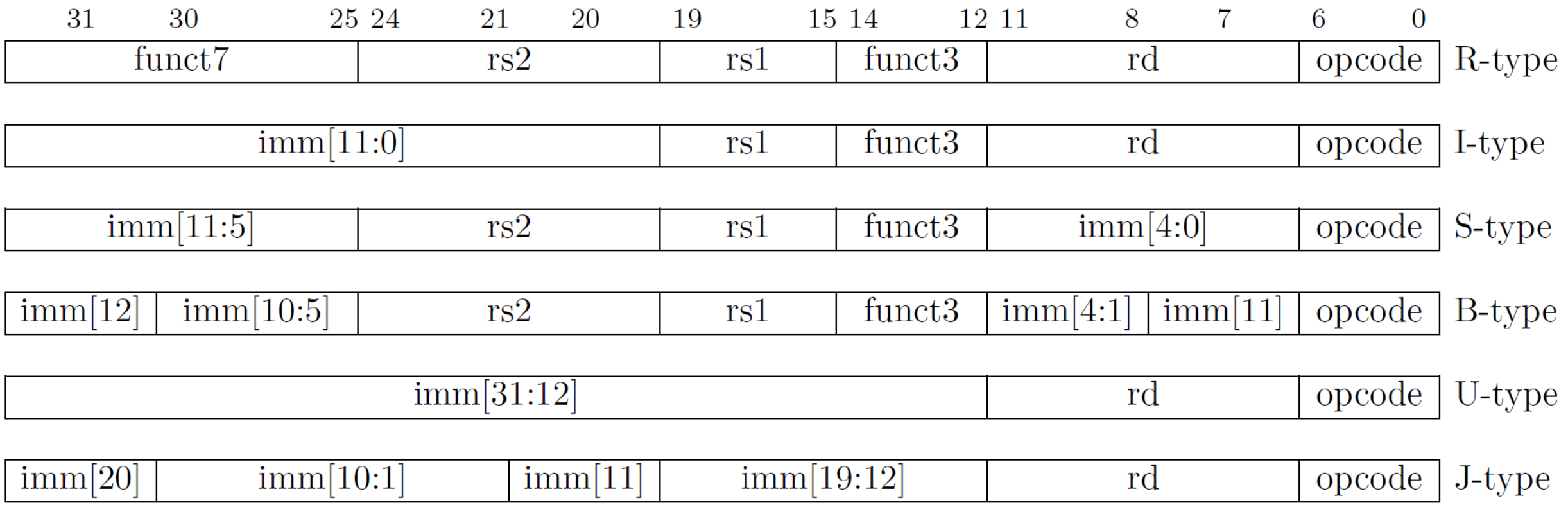
Расширения базовых наборов команд:

M	добавляет инструкции умножения, деления и вычисления остатка от деления целых чисел
A	добавляет атомарные инструкции чтения/записи для синхронизации между аппаратными потоками
F	добавляет инструкции для работы с числами с плавающей точкой одинарной точности
D	добавляет инструкции для работы с числами с плавающей точкой двойной точности
Q	добавляет инструкции для работы с числами с плавающей точкой четверной точности
C	добавляет поддержку сжатия инструкций до 16 бит для сокращения занимаемого объема памяти

Типы RISC-V инструкций

R-типа:	арифметические и логические операции над двумя регистрами с записью результата в третий (`add`, `xor`, `mul`)
I-типа:	инструкции с 12-битным непосредственным операндом (`addi`, `lw`, `slli`, `slti`)
S-типа:	инструкции записи в память (`sb`, `sh`, `sw`)
B-типа:	инструкции ветвления (`beq`, `bne`, `blt`, `bge`)
U-типа:	инструкции с 20-битным непосредственным операндом (`lui`, `auipc`)
J-типа:	единственная инструкция (`jal`), осуществляющая безусловный переход

Формат инструкций RISC-V



Непосредственные операнды (immediates)

Примеры:

`addi x5, x6, 4` – записать в регистр x5 сумму значения из регистра x6 и числа 4 (x5 = x6 + 4).

`slli x7, x8, 2` – записать в регистр x7 значения из регистра x8 сдвинутое влево на 2 (x7 = x8 ≪ 2).

`blt x9, x10, 60` – совершить условный переход по адресу PC + 60, если значение из регистра x9 меньше, чем из x10.

Для кодирования констант длиннее 12 бит используются команды U-типа с 20-битным операндом.
Например, требуется совершить переход по адресу PC + long_value:

`auipc x1, long_value[31:12]` – записать в x1 сумму PC и старших битов long_value

`jalr x1, x1, long_value[11:0]` – записать в PC сумму значения из x1 и младших битов long_value

Инструкции RV32I R-типа

Инструкция	Операция	opcode	funct3	funct7	Псевдокод	Дополнение
`add`	Сложение	`0110011`	`0x0`	`0x00`	`rd = rs1 + rs2`	
`sub`	Вычитание	`0110011`	`0x0`	`0x20`	`rd = rs1 - rs2`	
`xor`	Побитовое исключающее ИЛИ	`0110011`	`0x4`	`0x00`	`rd = rs1 ^ rs2`	
`or`	Побитовое ИЛИ	`0110011`	`0x6`	`0x00`	`rd = rs1 | rs2`	
`and`	Побитовое И	`0110011`	`0x7`	`0x00`	`rd = rs1 & rs2`	
`sll`	Логический сдвиг влево	`0110011`	`0x1`	`0x00`	`rd = rs1 << rs2`	
`srl`	Логический сдвиг вправо	`0110011`	`0x5`	`0x00`	`rd = rs1 >> rs2`	
`sra`	Арифметический сдвиг вправо	`0110011`	`0x5`	`0x20`	`rd = rs1 >> rs2`	
`slt`	1, если меньше, иначе – 0	`0110011`	`0x2`	`0x00`	`rd = (rs1 < rs2)?1:0`	сравнение знаковых чисел
`sltu`	1, если меньше, иначе – 0	`0110011`	`0x3`	`0x00`	`rd = (rs1 < rs2)?1:0`	сравнение беззнаковых чисел

Инструкции RV32I I-типа

Инструкция	Операция	opcode	funct3	funct7	Псевдокод	Дополнение
`addi`	Сложение с константой	`0010011`	`0x0`		`rd = rs1 + imm`	
`xori`	Побитовое исключающее ИЛИ с константой	`0010011`	`0x4`		`rd = rs1 ^ imm`	
`ori`	Побитовое ИЛИ с константой	`0010011`	`0x6`		`rd = rs1 | imm`	
`andi`	Побитовое И с константой	`0010011`	`0x7`		`rd = rs1 & imm`	
`slli`	Логический сдвиг влево на значение константы	`0010011`	`0x1`	`0x00`	`rd = rs1 << imm`	
`srli`	Логический сдвиг вправо на значение константы	`0010011`	`0x5`	`0x00`	`rd = rs1 >> imm`	
`srai`	Арифметический сдвиг вправо на значение константы	`0010011`	`0x5`	`0x20`	`rd = rs1 >> imm`	
`slti`	1, если меньше, иначе -- 0 (сравнение с константой)	`0010011`	`0x2`		`rd = (rs1 < imm)?1:0`	сравнение знаковых чисел
`sltiu`	1, если меньше, иначе -- 0 (сравнение с константой)	`0010011`	`0x3`		`rd = (rs1 < imm)?1:0`	сравнение беззнаковых чисел

Инструкции RV32I I-типа

Инструкция	Операция	opcode	funct3	funct7	Псевдокод	Дополнение
`lb`	Чтение 1 байта из памяти	`0000011`	`0x0`		`rd = M[rs1+imm][0:7]`	знаковые данные
`lh`	Чтение 2 байт из памяти	`0000011`	`0x1`		`rd = M[rs1+imm][0:15]`	знаковые данные
`lw`	Чтение 4 байт из памяти	`0000011`	`0x2`		`rd = M[rs1+imm][0:31]`	знаковые данные
`lbu`	Чтение 1 байта из памяти	`0000011`	`0x4`		`rd = M[rs1+imm][0:7]`	беззнаковые данные
`lhu`	Чтение 2 байт данных из памяти	`0000011`	`0x5`		`rd = M[rs1+imm][0:15]`	беззнаковые данные
`jalr`	Безусловный переход	`1100111`	`0x0`		`rd = PC+4; PC = rs1 + imm`	
`ecall`	Передача управления ОС	`1110011`	`0x0`	`0x00`		immediate = 0x000
`ebreak`	Передача управления отладчику	`1110011`	`0x0`	`0x00`		immediate = 0x001

Инструкции RV32I S-типа

Инструкция	Операция	opcode	funct3	funct7	Псевдокод	Дополнение
`sb`	Запись 1 байта в память	`0100011`	`0x0`		`M[rs1+imm][0:7] = rs2[0:7]`	
`sh`	Запись 2 байтов в память	`0100011`	`0x1`		`M[rs1+imm][0:15] = rs2[0:15]`	
`sw`	Запись 4 байтов в память	`0100011`	`0x2`		`M[rs1+imm][0:31] = rs2[0:31]`	

Инструкции RV32I B-типа

Инструкция	Операция	opcode	funct3	Псевдокод	Дополнение
`beq`	Условный переход, если ==	`1100011`	`0x0`	`if(rs1 == rs2) PC += imm`	
`bne`	Условный переход, если !=	`1100011`	`0x1`	`if(rs1 != rs2) PC += imm`	
`blt`	Условный переход, если <	`1100011`	`0x4`	`if(rs1 <  rs2) PC += imm`	сравнение знаковых чисел
`bge`	Условный переход, если ≥	`1100011`	`0x5`	`if(rs1 >= rs2) PC += imm`	сравнение знаковых чисел
`bltu`	Условный переход, если <	`1100011`	`0x6`	`if(rs1 <  rs2) PC += imm`	сравнение беззнаковых чисел
`bgeu`	Условный переход, если ≥	`1100011`	`0x7`	`if(rs1 >= rs2) PC += imm`	сравнение беззнаковых чисел

Инструкции RV32I U-типа

Инструкция	Операция	opcode	Псевдокод	Дополнение
`lui`	Сохранение константы в регистр	`0110111`	`rd = imm << 12`	
`auipc`	Сложение счетчика команд и константы	`0010111`	`rd = PC + (imm << 12)`	

Инструкции RV32I J-типа

Инструкция	Операция	opcode	Псевдокод	Дополнение
`jal`	Безусловный переход по адресу относительно текущего счетчика команд	`1101111`	`rd = PC+4; PC += imm`	

Инструкции RV32M

Инструкция	Операция	opcode	funct3	funct7	Псевдокод	Дополнение
`mul`	Умножение, младшие 32 бита	`0110011`	`0x0`	`0x01`	`rd = (rs1 * rs2)[31:0]`	
`mulh`	Умножение, старшие 32 бита	`0110011`	`0x1`	`0x01`	`rd = (rs1 * rs2)[63:32]`	оба операнда знаковые
`mulsu`	Умножение, старшие 32 бита	`0110011`	`0x2`	`0x01`	`rd = (rs1 * rs2)[63:32]`	первый операнд знаковый, второй – беззнаковый
`mulu`	Умножение, старшие 32 бита	`0110011`	`0x3`	`0x01`	`rd = (rs1 * rs2)[63:32]`	оба операнда беззнаковые
`div`	Деление	`0110011`	`0x4`	`0x01`	`rd = rs1 / rs2`	
`divu`	Деление	`0110011`	`0x5`	`0x01`	`rd = rs1 / rs2`	
`rem`	Вычисление остатка от деления	`0110011`	`0x6`	`0x01`	`rd = rs1 % rs2`	
`remu`	Вычисление остатка от деления	`0110011`	`0x7`	`0x01`	`rd = rs1 % rs2`	

Расширение набора команд RV32M включает в себя только команды R-типа.

Инструкции RV32F

Инструкция	Операция	Тип	rs2	opcode	funct3	funct7	Пояснение
fmadd.s	Flt Fused Mul-Add	R		1000011	rm	[rs3, 00]	rd = rs1 * rs2 + rs3
fmsub.s	Flt Fused Mul-Sub	R		1000111	rm	[rs3, 00]	rd = rs1 * rs2 - rs3
fnmadd.s	Flt Neg Fused Mul-Add	R		1001111	rm	[rs3, 00]	rd = -rs1 * rs2 + rs3
fnmsub.s	Flt Neg Fused Mul-Sub	R		1001011	rm	[rs3, 00]	rd = -rs1 * rs2 - rs3
fadd.s	Flt Add	R		1010011	rm	0x00	rd = rs1 + rs2
fsub.s	Flt Sub	R		1010011	rm	0x04	rd = rs1 - rs2
fmul.s	Flt Mul	R		1010011	rm	0x08	rd = rs1 * rs2
fdiv.s	Flt Div	R		1010011	rm	0x0C	rd = rs1 / rs2
fsqrt.s	Flt Square Root	R	00000	1010011	rm	0x2C	rd = sqrt(rs1)
fsgnj.s	Flt Sign Injection	R		1010011	0x0	0x10	rd = abs(rs1) * sgn(rs2)
fsgnjn.s	Flt Sign Neg Injection	R		1010011	0x1	0x10	rd = abs(rs1) * -sgn(rs2)
fsgnjx.s	Flt Sign Xor Injection	R		1010011	0x2	0x10	rd = rs1 * sgn(rs2)
fmin.s	Flt Minimum	R		1010011	0x0	0x14	rd = min(rs1, rs2)
fmax.s	Flt Maximum	R		1010011	0x1	0x14	rd = max(rs1, rs2)

Инструкции RV32F

Инструкция	Операция	Тип	rs2	opcode	funct3	funct7	Пояснение
`flw`	Flt Load Word	I		`0000111`	`0x2`		`rd = M[rs1 + imm]`
`fsw`	Flt Store Word	S		`0100111`	`0x2`		`M[rs1 + imm] = rs2`
`fcvt.w.s`	Flt Convert to Int	R	`00000`	`1010011`	`rm`	`0x60`	`rd = (int32_t) rs1`
`fcvt.wu.s`	Flt Convert to Int	R	`00001`	`1010011`	`rm`	`0x60`	`rd = (uint32_t) rs1`
`fmv.x.s`	Move Float to Int	R	`00000`	`1010011`	`0x0`	`0x70`	`rd = *((int*) &rs1)`
`fmv.s.x`	Move Int to Float	R	`00000`	`1010011`	`0x0`	`0x78`	`rd = *((float*) &rs1)`
`feq.s`	Float Equality	R		`1010011`	`0x2`	`0x50`	`rd = (rs1 == rs2) ? 1 : 0`
`flt.s`	Float Less Than	R		`1010011`	`0x1`	`0x50`	`rd = (rs1 <  rs2) ? 1 : 0`
`fle.s`	Float Less / Equal	R		`1010011`	`0x0`	`0x50`	`rd = (rs1 <= rs2) ? 1 : 0`
`fclass.s`	Float Classify	R	`00000`	`1010011`	`0x1`	`0x70`	`rd = 0..9`

Округление результатов вычисления команд RV32F

RM – Rounding Mode (метод округления). Значение RM помещается в поле func3 расширения набора команд RV32F R-типа.

Значение	Метод округления
000	Округление к ближайшему целому, если 0,5 – к ближайшему четному
001	Округление в сторону нуля
010	Округление вниз (в сторону $-\infty$)
011	Округление вниз (в сторону $+\infty$)
100	Округление к ближайшему целому, если 0,5 – к наибольшему по модулю значению
101	Некорректное значение
110	Некорректное значение
111	Динамический режим (режим задается в статусном регистре)

Псевдоинструкции RV32I

Псевдоинструкция	Декодирование	Пояснение
`nop`	`addi x0, x0, 0`	No operation (нет операции)
`la rd, symbol`	`auipc rd, symbol[31:12]` `addi rd, rd, symbol[11:0]`	Загрузить адрес
`li rd, immediate`	Различные варианты	Загрузить константу
`mv rd, rs`	`addi rd, rs, 0`	Копировать значение
`not rd, rs`	`xori rd, rs, -1`	Обратный код
`neg rd, rs`	`sub rd, x0, rs`	Дополнительный код
`seqz rd, rs`	`sltiu rd, rs, 1`	Установить, если 0
`snez rd, rs`	`sltu rd, x0, rs`	Установить, если не 0
`sltz rd, rs`	`slt rd, rs, x0`	Установить, если < 0
`sgtz rd, rs`	`slt rd, x0, rs`	Установить, если > 0
`fmv.s rd, rs`	`fsgnj.s rd, rs, rs`	Копировать число одинарной точности
`fabs.s rd, rs`	`fsgnjx.s rd, rs, rs`	Модуль числа одинарной точности
`fneg.s rd, rs`	`fsgnjn.s rd, rs, rs`	Противоположное числу одинарной точности
`fmv.d rd, rs`	`fsgnj.d rd, rs, rs`	Копировать число двойной точности
`fabs.d rd, rs`	`fsgnjx.d rd, rs, rs`	Модуль числа двойной точности
`fneg.d rd, rs`	`fsgnjn.d rd, rs, rs`	Противоположное числу двойной точности

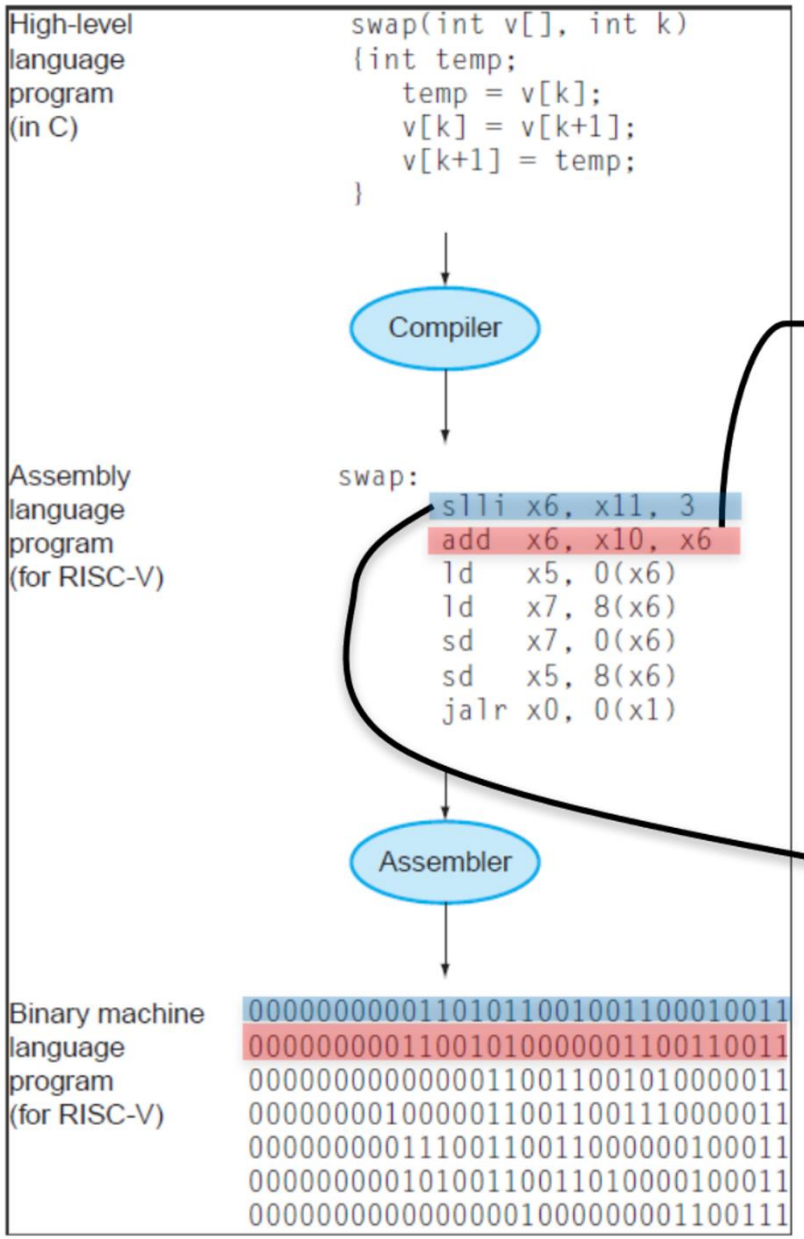
Псевдоинструкции RV32I

Псевдоинструкция	Декодирование	Пояснение
`beqz rs, offset`	`beq rs, x0, offset`	Переход, если 0
`bnez rs, offset`	`bne rs, x0, offset`	Переход, если не 0
`blez rs, offset`	`bge x0, rs, offset`	Переход, если ≤ 0
`bgez rs, offset`	`bge rs, x0, offset`	Переход, если ≥ 0
`bltz rs, offset`	`blt rs, x0, offset`	Переход, если < 0
`bgtz rs, offset`	`blt x0, rs, offset`	Переход, если > 0
`bgt rs, rt, offset`	`blt rt, rs, offset`	Переход, если $>$
`ble rs, rt, offset`	`bge rt, rs, offset`	Переход, если $\leq$
`bgtu rs, rt, offset`	`bltu rt, rs, offset`	Переход, если $>$, значения беззнаковые
`bleu rs, rt, offset`	`bgeu rt, rs, offset`	Переход, если $\leq$, значения беззнаковые

Псевдоинструкции RV32I

Псевдоинструкция	Декодирование	Пояснение
`j offset`	`jal x0, offset`	Переход без записи адреса в rd
`jal offset`	`jal x1, offset`	Безусловный переход
`jr rs`	`jalr x0, rs, 0`	Переход по адресу rs без записи адреса в rd
`jalr rs`	`jalr x1, rs, 0`	Переход по адресу rs
`ret`	`jalr x0, x1, 0`	Возврат из подпрограммы
`call offset`	`auipc x1, offset[31:12]` `jalr x1, x1, offset[11:0]`	Вызов подпрограммы
`tail offset`	`auipc x6, offset[31:12]` `jalr x0, x6, offset[11:0]`	“Tail call” – вызов текущей подпрограммы перед ее завершением

Представление ассемблерных инструкций



- RISC-V инструкция R-типа

funct7	rs2	rs1	funct3	rd	opcode
7 бит	5 бит	5 бит	3 бита	5 бит	7 бит

`add x6, x10, x6`

0	6	10	0	6	51

0000000	00110	01010	000	00110	0110011

$0000\ 0000\ 0110\ 0101\ 0000\ 0011\ 0011\ 0011_{two} = 00650333_{16}$

- RISC-V инструкция I-типа

immediate	rs1	funct3	rd	opcode
12 бит	5 бит	3 бита	5 бит	7 бит

Инструкция логического сдвига влево:
rs1 – сдвигаемый операнд
immediate – значение сдвига (первые 5 бит)

Директивы ASM файла

- Начинаются с точки (‘.’)
- Указывают на то, как следует транслировать программу и не транслируются в инструкции
- Некоторые примеры:

`.equ`	– объявление константы
`.word`	– объявление массива 32-битных слов
`.string`	– объявление строки
`.text`	– указание на раздел с исполняемым кодом, read-only
`.data`	– указание на раздел с данными, read-write
`.rodata`	– указание на раздел с константами, read-only

Пример содержимого секции .data:

```
.data
  num1: .word 1
  num2: .word 3
  arr1: .word 5 7 1
  s1: .string “AB\n“
  # comment
```

Директивы ASM файла

`__start` – это “label”, или символ, значение которого равно адресу в памяти ассемблерного кода, который начинается после объявления `__start`.

Директива `.globl` делает символ видимым для компоновщика (“linker”). Директива указывает на символ `__start`, так как требуется сообщить компоновщику, чтобы тот поместил код, на который указывает символ, в корректное место в памяти.

Использование директивы .globl:

```
.globl __start

.text
__start:

  # исполняемый код
```

Пример 1. Вычисление суммы элементов массива

Задача:

В памяти лежит массив длины 5.

Найти сумму элементов массива.

```
.globl __start

.data
  array: .word 1, 2, 3, 4, 5 # A=[1, 2, 3, 4, 5];

.text
__start:
  la x9, array          # x9=&A[0]
  addi x10, x0, 0       # sum=0
  addi x11, x0, 0       # i=0
Loop:
  lw x12, 0(x9)         # x12=A[i]
  add x10, x10, x12     # sum+=
  addi x9, x9, 4        # &A[i++]
  addi x11, x11, 1      # i++
  addi x13, x0, 5       # x13=5
  blt x11, x13, Loop
  nop
  call Exit

Exit:
  tail Exit
```

Пример 2. Сортировка элементов массива

Задача:
В памяти лежит массив длины 8. Отсортировать по убыванию и записать с другую область памяти.

1) Описываем начальные данные

```
.globl __start

.data
  array: .word 1, 4, 8, 7, 2, 2, 9, 3

.rodata
  len: .word 8

.text
__start:
  la x9, array     # x9 = &A[0]
  mv x11, x0       # i = 0
  lw t1, len
  mv x15, t1       # x15 = len

  jal COPY         # вызов подпрограммы для
                   # копирования текущего массива
                   # в соседние адреса
  ...
```

Пример 2. Сортировка элементов массива

Задача:
В памяти лежит массив длины 8. Отсортировать по убыванию и записать с другую область памяти.

2) Описываем подпрограмму копирования массива по новому адресу

```
COPY:
  lw t1, len                # t1 = len
  slli t0, t1, 2
  add t0, x9, t0            # t0 = &A[0] + 8 << 2
  mv x10, x9                # x10 = &A[0]
  mv x11, x0                # i = 0
  mv x9, t0                 # x9 = t0 (new_addr)
  CP_Loop:
    lw t3, 0(x10)           # t3 = A[i]
    sw t3, 0(t0)            # запись A[i] по адресу
t0
    addi x10, x10, 4        # old_addr++
    addi t0, t0, 4          # new_addr++
    addi x11, x11, 1        # i++
    blt x11, x15, CP_Loop
  ret
```

Пример 2. Сортировка элементов массива

Задача:

В памяти лежит массив длины 8. Отсортировать по убыванию и записать с другую область памяти.

3) Определяем два цикла:

```
for (int i = 0; i < SIZE; i++) {
    max = A[i];
    max_ind = i;
    for (int j = i; j < SIZE; j++) {
      if (max < B[j]) {
        goto MAX;
      }
    }
```

```
  ...
  mv x11, x0                # i = 0
Loop1:
  mv x13, x11               # max_ind = i
  mv x14, x11               # j = i
  mv x17, x11
  slli x17, x17, 2
  add x17, x9, x17
  lw x12, 0(x17)            # max = A[i]
  Loop2:
    mv x17, x14
    slli x17, x17, 2
    add x17, x9, x17
    lw x16, 0(x17)          # x16 = A[j]
    blt x12, x16, MAX       # if (max < x16)
R:  addi x14, x14, 1        # j++
    blt x14, x15, Loop2

    ...

    # перестановка элементов массива
```

Пример 2. Сортировка элементов массива

Задача:
В памяти лежит массив длины 8. Отсортировать по убыванию и записать с другую область памяти.

3) Описываем перестановку элементов массива и подпрограмму MAX

```
    # перестановка элементов массива
    slli x13, x13, 2
    add x13, x9, x13
    lw t1, 0(x13)           # t1 = A[max_ind]
    slli t2, x11, 2
    add t2, x9, t2
    lw t3, 0(t2)            # t3 = A[i]
    sw t3, 0(x13)           # A[max_ind] = t3
    sw t1, 0(t2)            # A[i] = t1
    addi x11, x11, 1        # i++

    blt x11, x15, Loop1
    nop
    call Exit

MAX:
  mv x12, x16          # max = x16
  mv x13, x14          # max_ind = j
  j R

Exit:
  tail Exit
```

Пример 2. Сортировка элементов массива

```
# полный код программы
.globl __start

.data
  array: .word 1,4,8,7,2,2,9,3

.rodata
  len: .word 8

.text
__start:
  la x9, array
  mv x11, x0
  lw t1, len
  mv x15, t1

  jal COPY

  mv x11, x0
Loop1:
  mv x13, x11
  mv x14, x11
  mv x17, x11
  slli x17, x17, 2
  add x17, x9, x17
  lw x12, 0(x17)
```

```
 Loop2:
    mv x17, x14
    slli x17, x17, 2
    add x17, x9, x17
    lw x16, 0(x17)

    blt x12, x16, MAX
R:  addi x14, x14, 1
    blt x14, x15, Loop2

    slli x13, x13, 2
    add x13, x9, x13
    lw t1, 0(x13)
    slli t2, x11, 2
    add t2, x9, t2
    lw t3, 0(t2)
    sw t3, 0(x13)
    sw t1, 0(t2)

    addi x11, x11, 1

    blt x11, x15, Loop1
    nop
    call Exit
```

```
MAX:
  mv x12, x16
  mv x13, x14
  j R

COPY:
  lw t1, len
  slli t0, t1, 2
  add t0, x9, t0
  mv x10, x9
  mv x11, x0
  mv x9, t0

  CP_Loop:
    lw t3, 0(x10)
    sw t3, 0(t0)
    addi x10, x10, 4
    addi t0, t0, 4
    addi x11, x11, 1
    blt x11, x15, CP_Loop
  ret

Exit:
  tail Exit
```