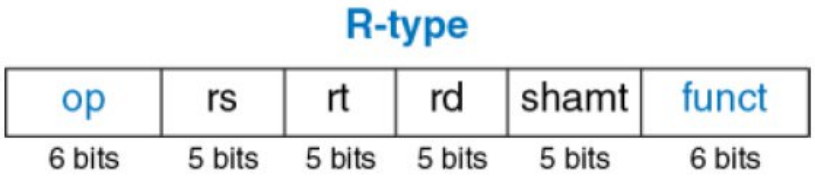
Архитектура MIPS / регистры (1)

Название	Номер	Назначение
$0	0	Константный нуль
$at	1	Временный регистр для нужд ассемблера
$v0–$v1	2–3	Возвращаемые функциями значения
$a0–$a3	4–7	Аргументы функций
$t0–$t7	8–15	Временные переменные
$s0–$s7	16–23	Сохраняемые переменные
$t8–$t9	24–25	Временные переменные
$k0–$k1	26–27	Временные переменные операционной системы (ОС)
$gp	28	Глобальный указатель (англ.: global pointer)
$sp	29	Указатель стека (англ.: stack pointer)
$fp	30	Указатель кадра стека (англ.: frame pointer)
$ra	31	Регистр адреса возврата из функции

- 32 регистра **общего** назначения по 4 байта (32 бита)
- адрес текущей команды хранится в **специальном** 32-х битном регистре, который называют счётчиком команд (англ.: program counter, PC)
- существуют другие регистры **специального** назначения (например, для хранения 64-х битного результата умножения или остатка отделения деления)

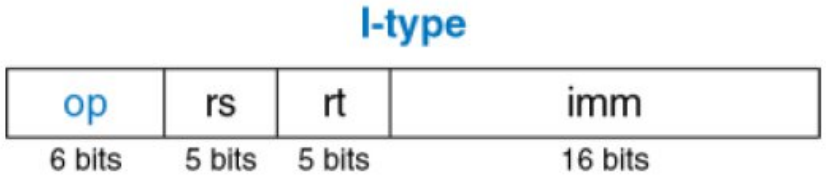
Архитектура MIPS / инструкции (2)

- длина всех инструкций 32 бита
- существуют 3 формата инструкий:
  - **R (register-type):** операнды - три регистра



пример: `add $t0, $s4, $s5`

  - **I (immediate):** операнды - два регистра и 16-ти битовая константа

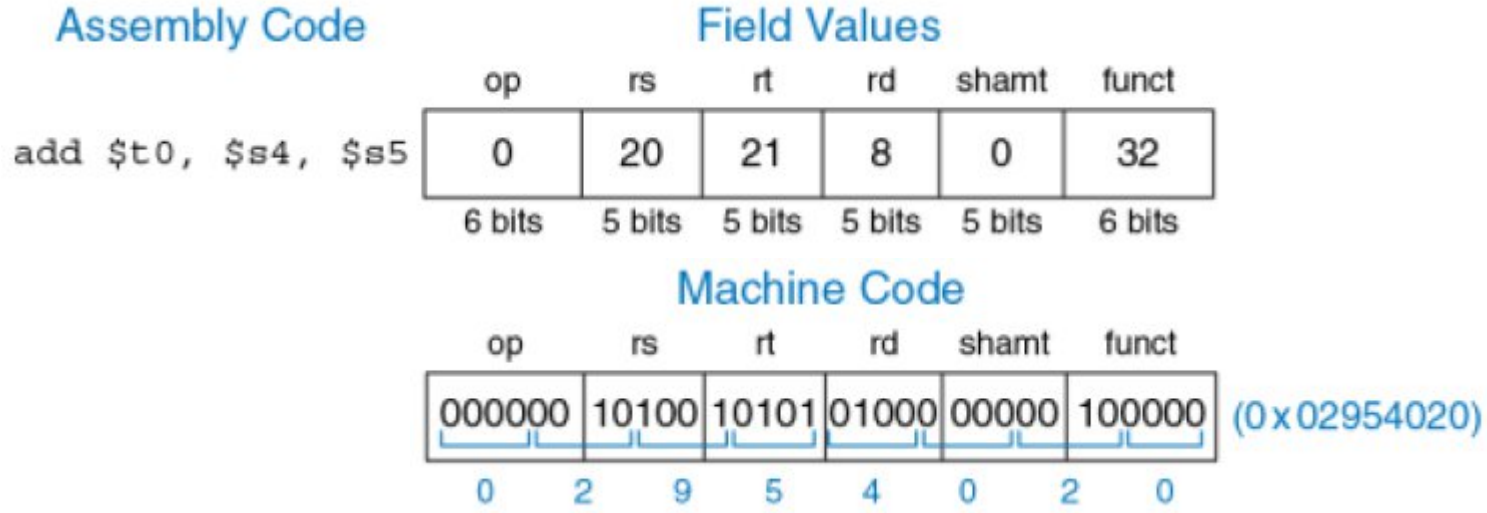


пример:
```
addi  $s0, $0, 4
lw    $s3,44($s1)
```

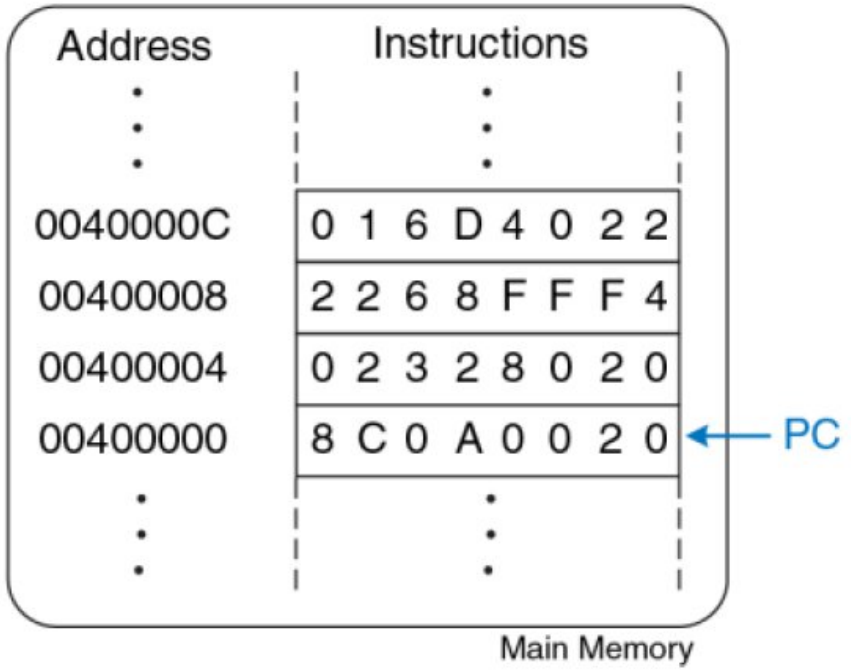
  - **J (jump):** операнд - 26-ти битовая константа (адрес перехода)

пример: `j target`

Архитектура MIPS / машинные коды и код ассемблера (3)



Assembly Code	Machine Code
`lw $t2, 32($0)`	`0x8C0A0020`
`add $s0, $s1, $s2`	`0x02328020`
`addi $t0, $s3, -12`	`0x2268FFF4`
`sub $t0, $t3, $t5`	`0x016D4022`



Архитектура MIPS / операции переходов (4)

Код на языке высокого уровня

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

Код на языке ассемблера MIPS

```
# $s0 = f, $s1 = g, $s2 = h, $s3 = i, $s4 = j
  bne $s3, $s4, else     # if i != j, branch to else
  add $s0, $s1, $s2      # if block: f = g + h
  j   L2                 # skip past the else block
else: <-------- (метка)
  sub $s0, $s0, $s3      # else block: f = f - i
L2: <-------- (метка)
```

Условный переход (**I тип инструкции**): `bne $s3, $s4, else`

При сборке в поле **imm** будет подставлено смещение относительно счётчика команд, и байт команды, помеченной меткой, будет PC’ = PC + 4 + imm * 4

Безусловный переход (**J тип инструкции**): `j  L2`

При сборке в поле **addr** будет подставлен абсолютный агрес команды после метки (точнее биты этого адреса с 3-го по 28-й)

Переход к микроархитектуре (5)

Для простоты ограничимся поддержкой следующего набора команд:

- **R (register-type):**

```
add     reg1, reg2, reg3     # reg1 = reg2 + reg3
sub     reg1, reg2, reg3     # reg1 = reg2 – reg3
and     reg1, reg2, reg3     # reg1 = reg2 & reg3
or      reg1, reg2, reg3     # reg1 = reg2 |  reg3
slt     reg1, reg2, reg3     # reg1 = reg2 < reg3 ? 1 : 0     (set less than)
```

- **I (immediate):**

```
lw      reg1, imm(reg2)      # чтение слова из памяти imm+reg2 в регистр reg1
sw      reg1, imm(reg2)      # запись слова из регистра reg1 в память imm+reg2
beq     reg1, reg2, target   # go to target if reg1 == reg2  (branch  if  equal)
addi    reg1, reg2, imm      # reg1 = reg2 + imm
```

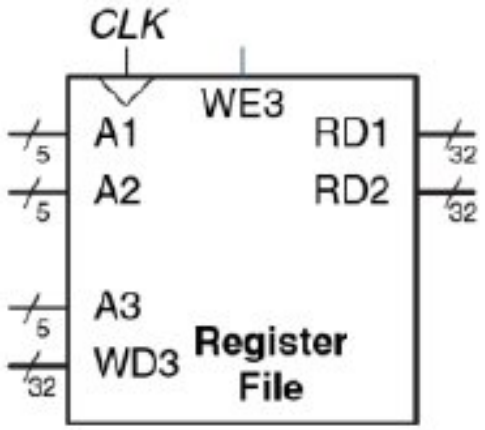
- **J (jump):**

```
j       target               # безусловный переход
```

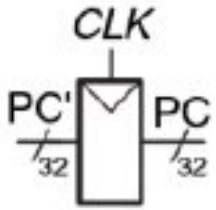
Процесс разработки проще организовать так: (1) начать с комбинационных схем, способных выполнять каждую отдельную команду; (2) объединить их в общий тракт данных; (3) определить таблицу истинности устройства управления трактом данных (главного дешифратора команд)

Микроархитектура / элементы памяти (6)

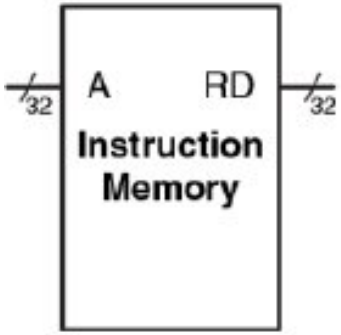
1. Файл регистров (32 регистра)



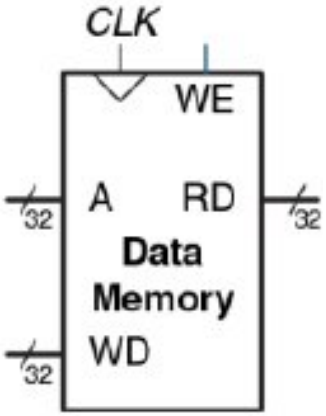
2. Счётчик команд



3. Память с инструкциями программы (только чтение)



4. Память с данными программы (чтение-запись)



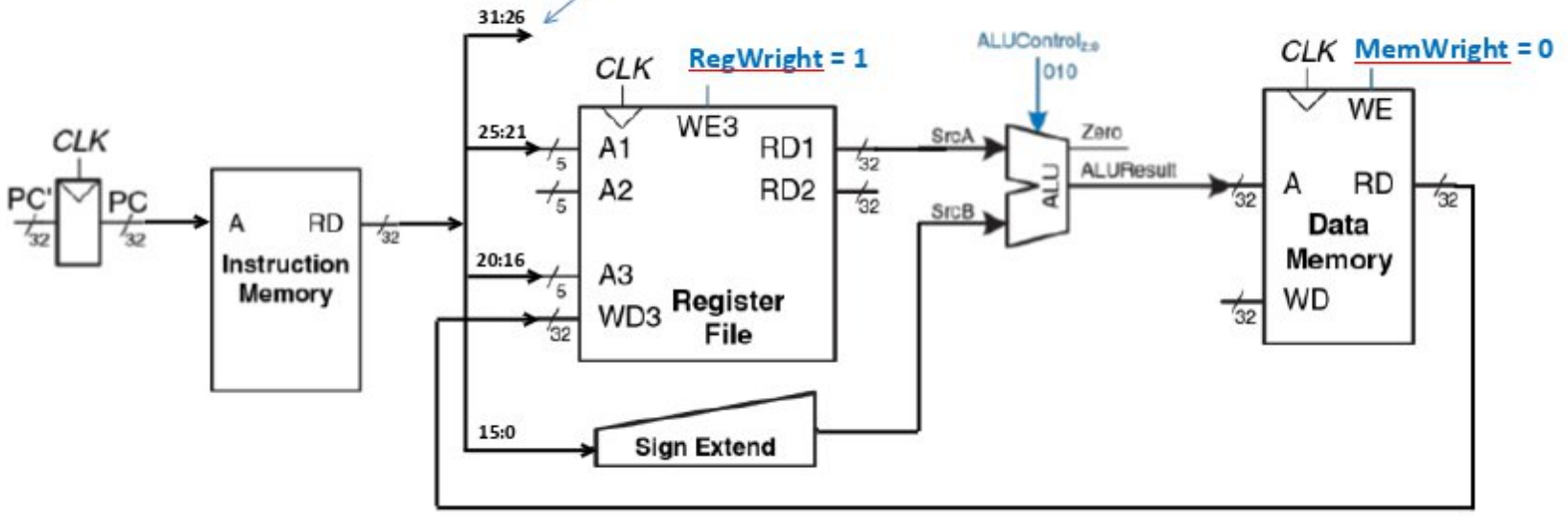
! Разделение внешней памяти на отдельные блоки (элементы схемы) для инструкций и данных сделано для упрощения микроархитектуры

Инструкция lw (загрузка слова из памяти в регистр) (7)

Рис. 6.8 **Формат команды типа *I***

op – тип операции
rs – базовый адрес, imm – смещение
rt – регистр-назначения

подаётся на вход управляющего устройства (УУ)



Сигналы на выходе УУ:
RegWrite = 1
ALUControl = 010
MemWrite = 0

Инструкция sw (сохранение слова из регистра в память) (8)

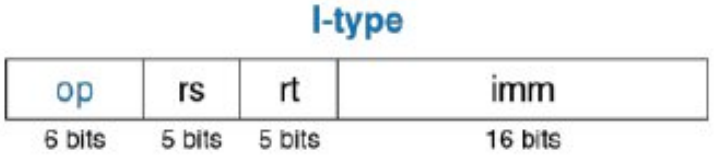
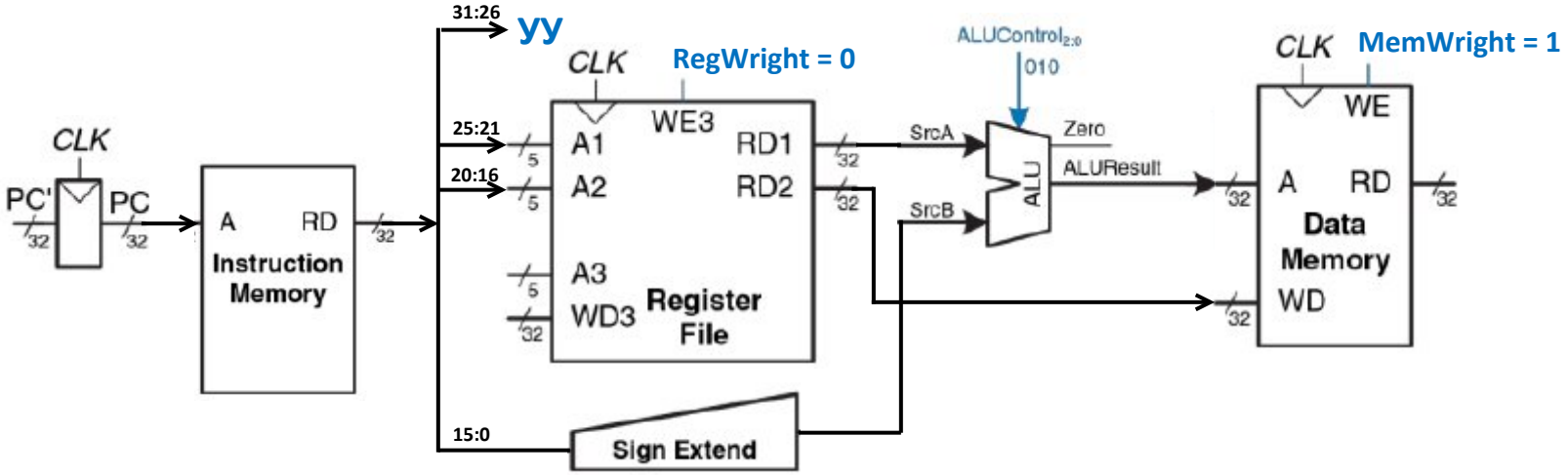


Рис. 6.8 Формат команды типа *I*

op – тип операции
rs – базовый адрес, imm – смещение
rt – регистр-источник

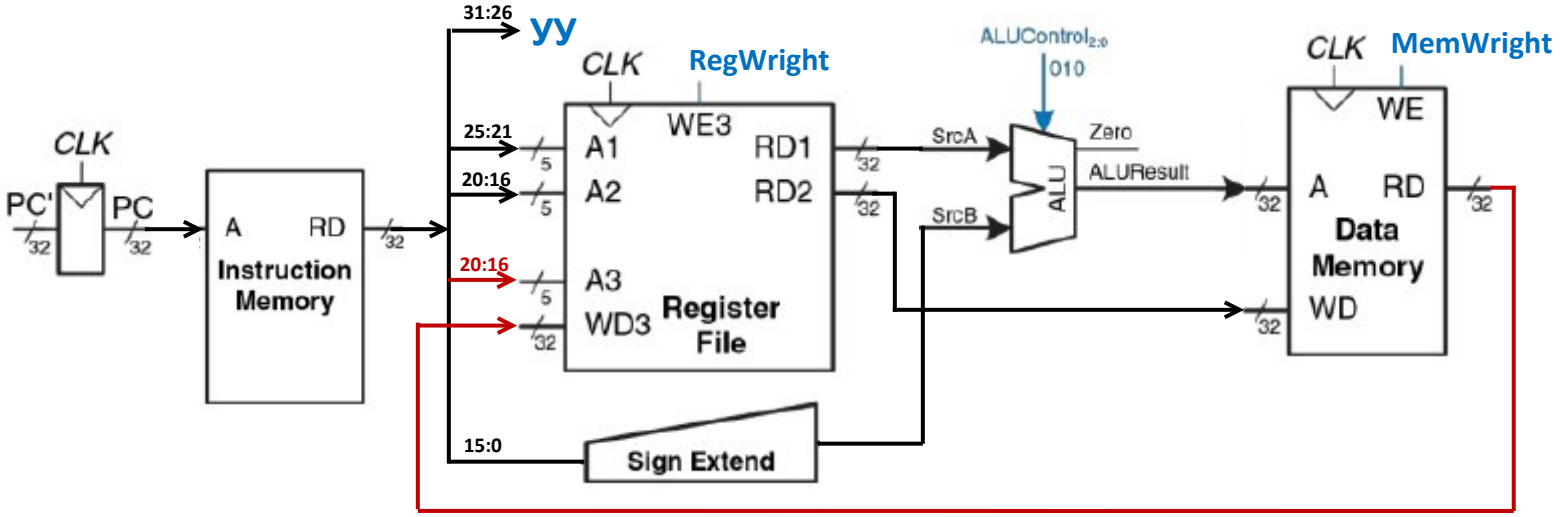


Сигнал на выходе УУ:
RegWrite = 0
ALUControl = 010
MemWrite = 1

Объединение схем для ls и sw (9)

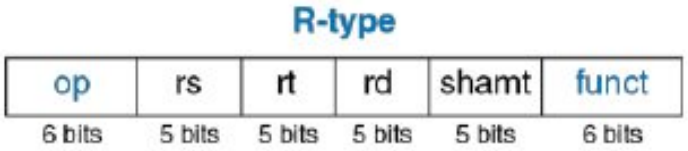
Рис. 6.8 Формат команды типа *I*

op – тип операции
rs – базовый адрес, imm – смещение
rt – регистр назначения/регистр-источник

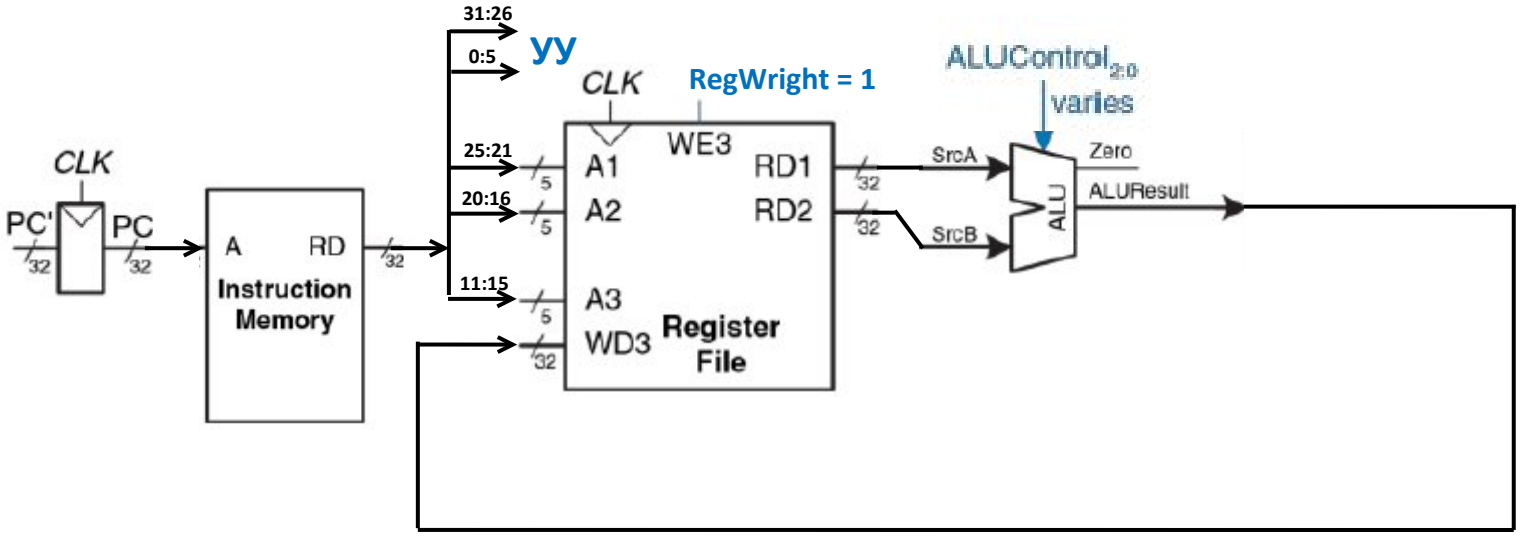


Сигнал на выходе УУ:
если RegWrite = 1 и MemWrite = 0 то это схема для команды lw
если RegWrite = 0 и MemWrite = 1 то это схема для команды sw

Инструкция add, sub, and, or, slt

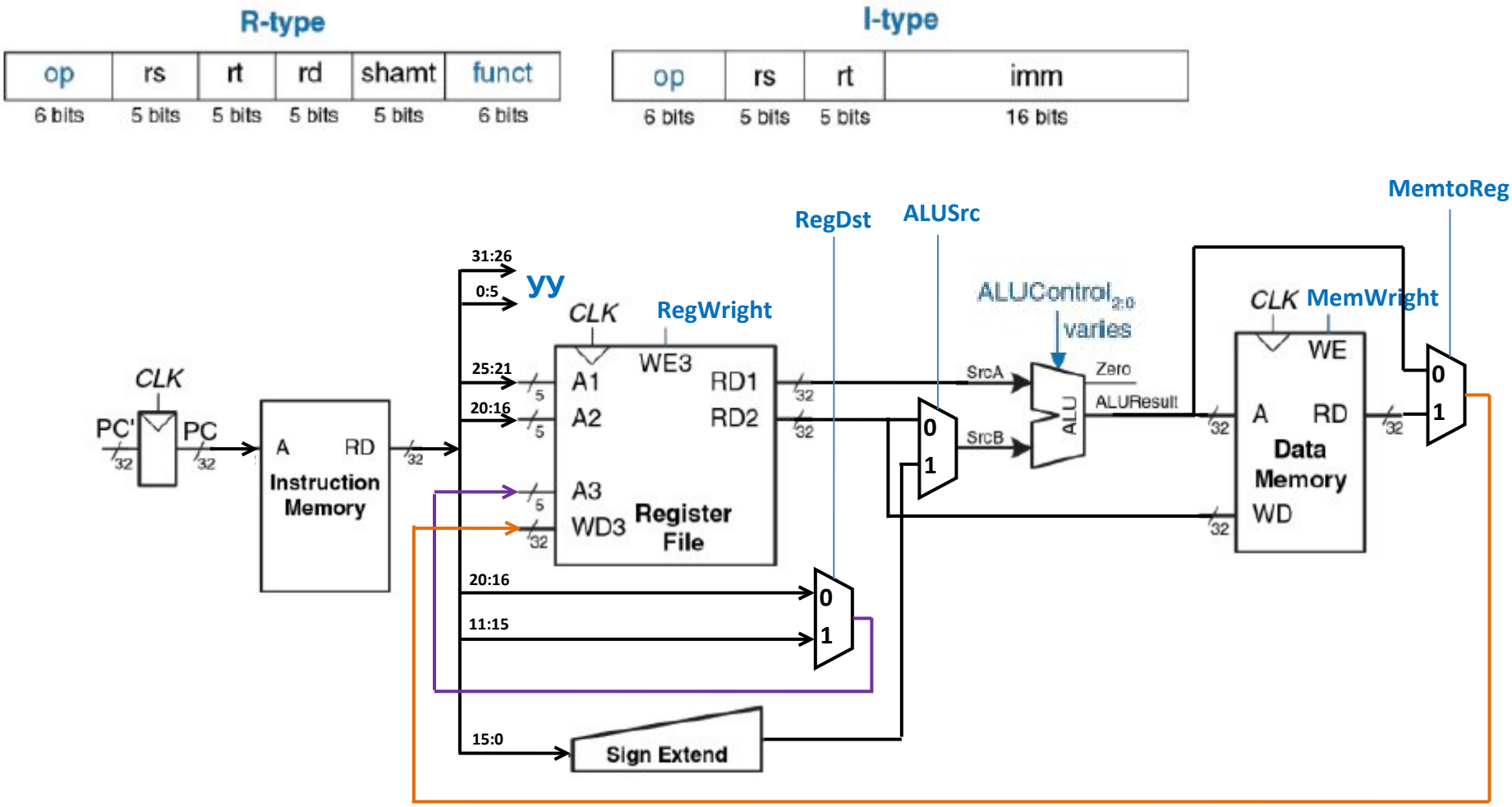


op и shamt = 0
rs и rt – коды регистров источников
rd – код регистра назначения



Сигнал на выходе УУ:
RegWrite = 1
ALUControl определяет тип операции

Объединение схем для инструкций R и I типов (11)

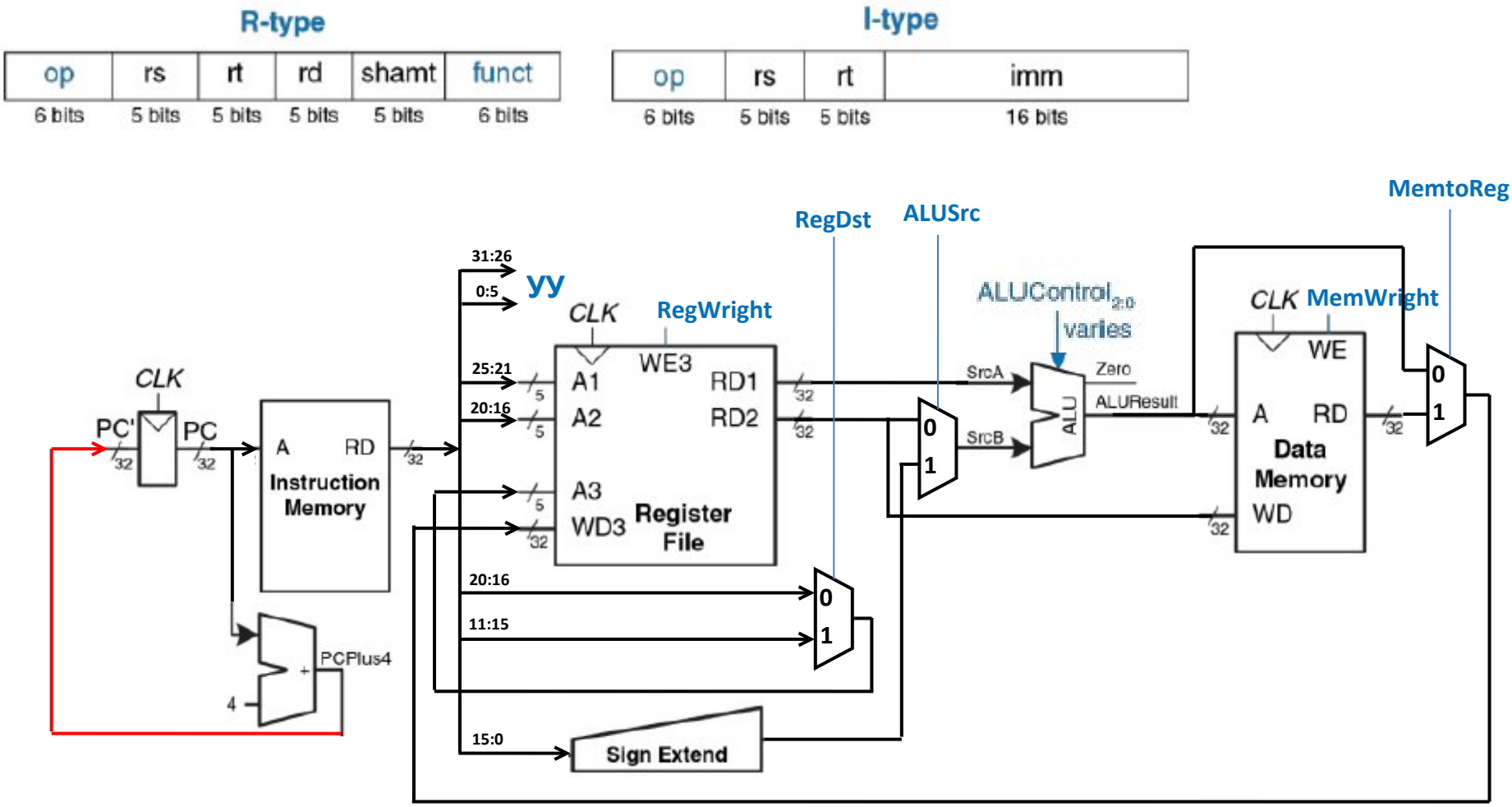


Мультиплексоры управляют распространением сигнала и тем, какой тип команды будет реализовываться:

RegDst = 1, ALUSrc = 0, MemtoReg = 1 – получаем схему для R (стр 10)

RegDst = 0, ALUSrc = 1, MemtoReg = 0 – получаем схему для I (стр 9)

Переход к следующей инструкции (12)

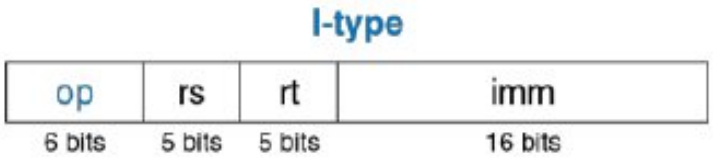


Мультиплексоры управляют распространением сигнала и тем, какой тип команды будет реализовываться:

RegDst = 1, ALUSrc = 0, MemtoReg = 1 – получаем схему для R (стр 10)

RegDst = 0, ALUSrc = 1, MemtoReg = 0 – получаем схему для I (стр 9)

Инструкция baq (условный переход)

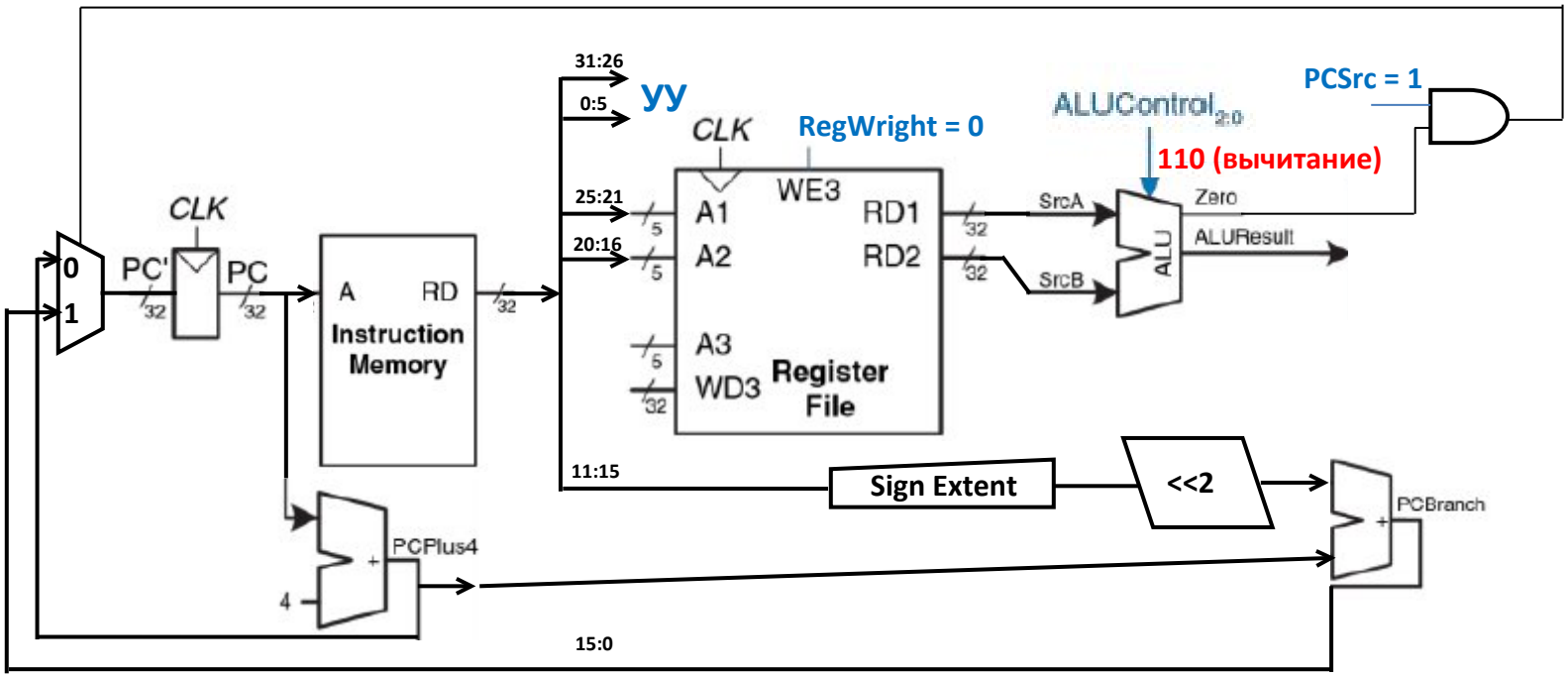


op – тип операции

rs, rt – регистры для сравнения

imm – сдвиг счётчика команд

PC' = PC + 4 + imm * 4



Сигнал на выходе УУ:

PCSrc = 1

RegWright = 0

ALUControl = 110 (вычитание)

Законченный (почти) однотактовый процессор (14)

- осталось построить таблицу истинности для управляющего устройства
- добавить команды **addi** и **j**

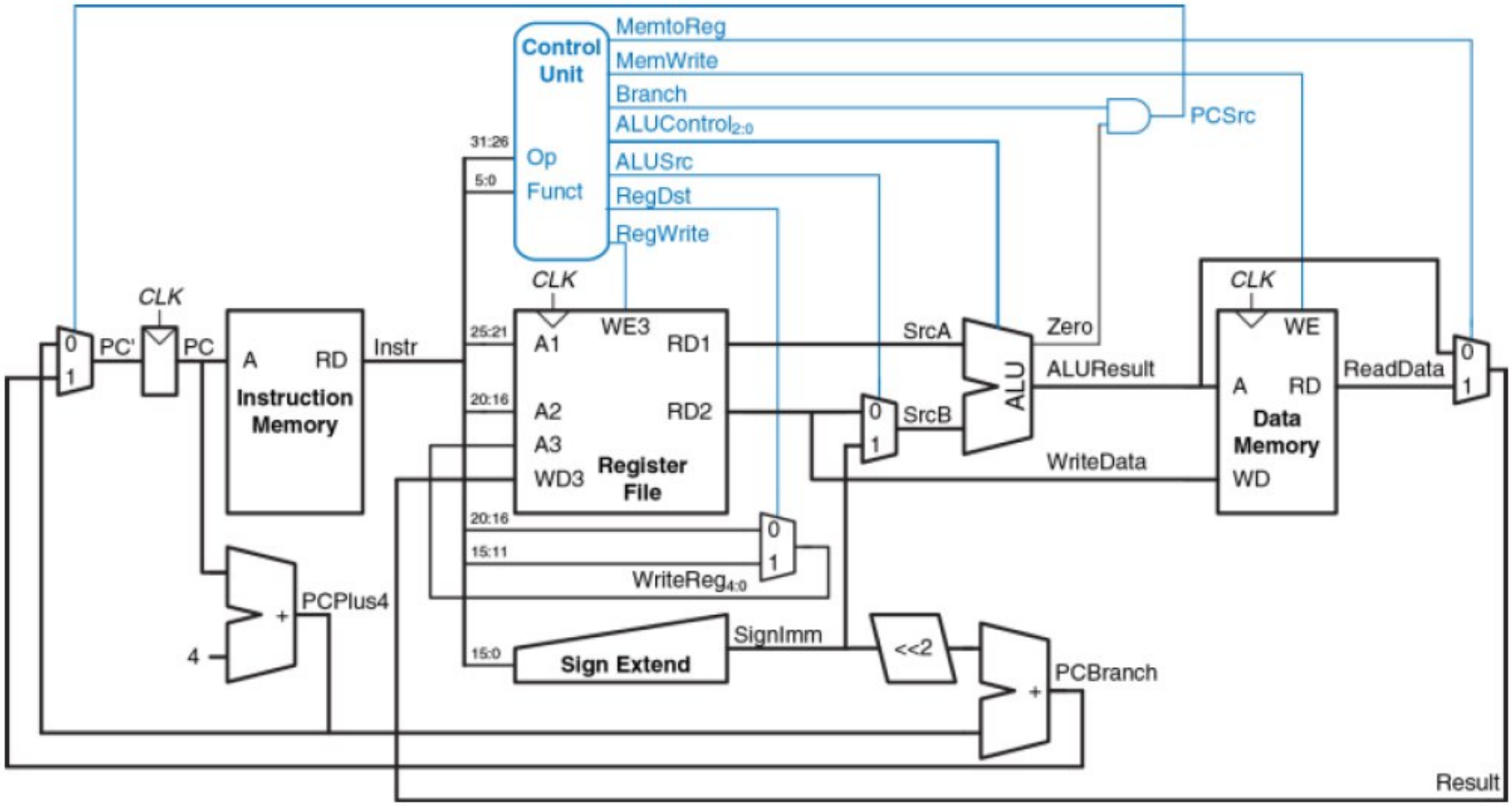


Таблица истинности дешифратора АЛУ (15)

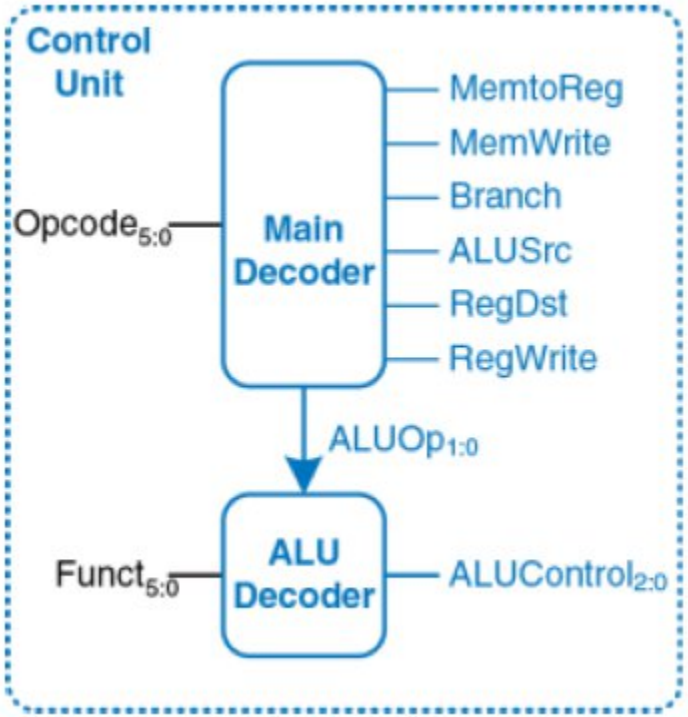


Табл. 7.1 **Расшифровка ALUOp**

ALUOp	Функция
00	Сложение
01	Вычитание
10	Определяется полем `funct`
11	Не используется

Табл. 7.2 **Таблица истинности дешифратора АЛУ**

ALUOp	funct	ALUControl
00	X	010 (сложение)
X1	X	110 (вычитание)
1X	100000 (`add`)	010 (сложение)
1X	100010 (`sub`)	110 (вычитание)
1X	100100 (`and`)	000 (логическое «И»)
1X	100101 (`or`)	001 (логическое «ИЛИ»)
1X	101010 (`slt`)	111 (установить, если меньше)

Таблица истинности основного дешифратора (16)

Табл. 7.3 Таблица истинности основного дешифратора

Команда	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp
Команды типа R	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	1	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01

Табл. 7.4 Таблица истинности основного дешифратора с поддержкой `addi`

Команда	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp
Команды типа R	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	1	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01
`addi`	001000	1	0	1	0	0	0	00

Законченный однотактовый процессор (17)

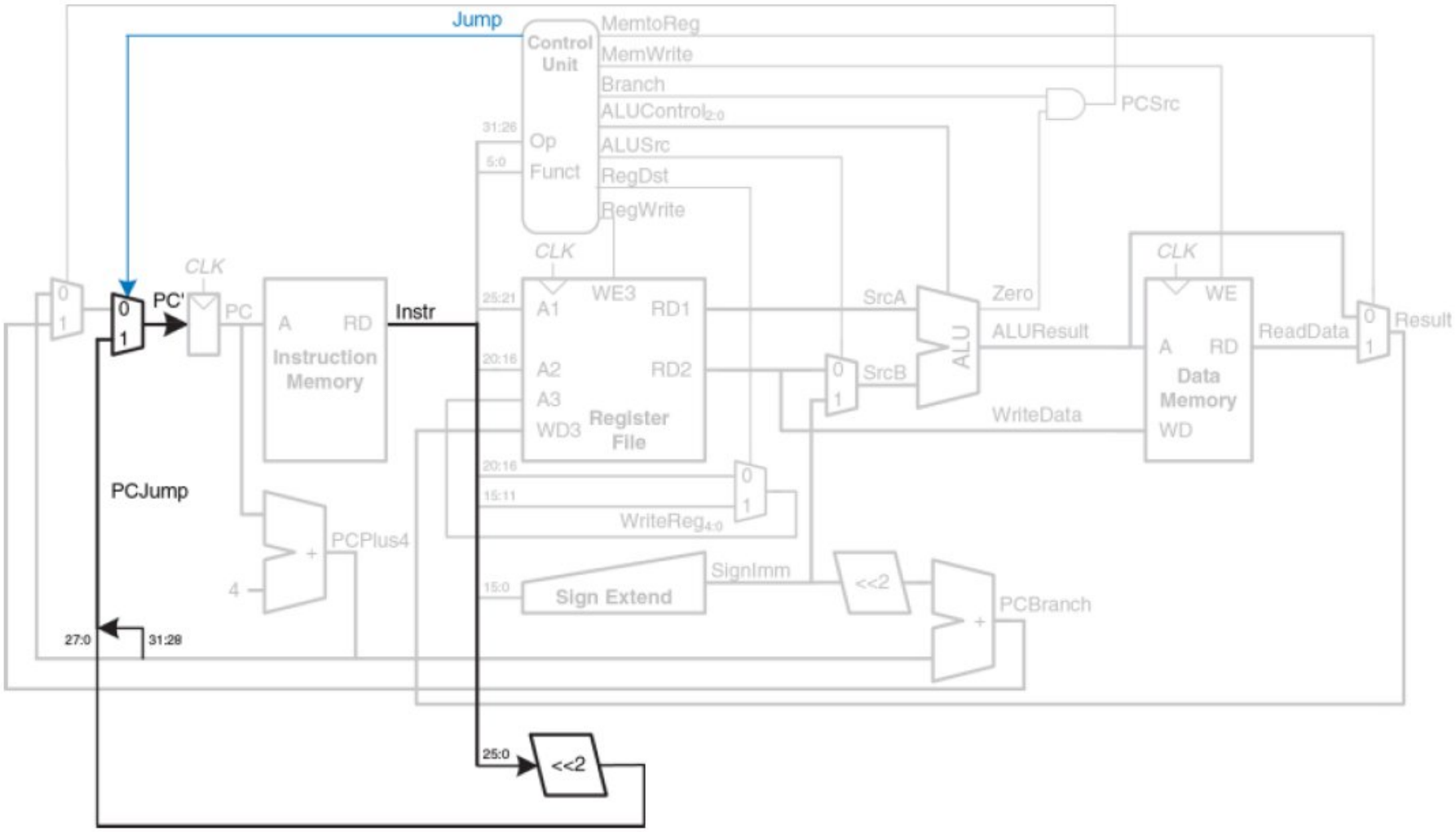


Рис. 7.14 Изменения в тракте данных для поддержки команды `j`

Финальная таблица истинности основного дешифратора (18)

Табл. 7.5 Таблица истинности основного дешифратора с поддержкой `j`

Команда	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp	Jump
Команды типа R	000000	1	1	0	0	0	0	10	0
`lw`	100011	1	0	1	0	0	1	00	0
`sw`	101011	0	X	1	0	1	X	00	0
`beq`	000100	0	X	0	1	0	X	01	0
`addi`	001000	1	0	1	0	0	0	00	0
`j`	000010	0	X	X	X	0	X	XX	1