Архитектура вычислительных систем

Лекция 10.

Яревский Е.А.

Кафедра вычислительной физики

Микроархитектура

Микроархитектура является связующим звеном между логическими схемами и архитектурой.

Микроархитектура описывает, как именно в процессоре расположены и соединены друг с другом регистры, АЛУ, конечные автоматы, блоки памяти и другие блоки, необходимые для реализации архитектуры.

У каждой архитектуры может быть много различных микроархитектур, обеспечивающих разное соотношение производительности, цены и сложности.

Компьютерная архитектура определяется набором команд и архитектурным состоянием. Архитектурное состояние процессора MIPS определяется содержимым счетчика команд (PC) и 32 видимых программисту регистров, поэтому любой процессор, реализующий архитектуру MIPS, вне зависимости от его микроархитектуры обязан иметь счетчик команд и ровно 32 регистра.

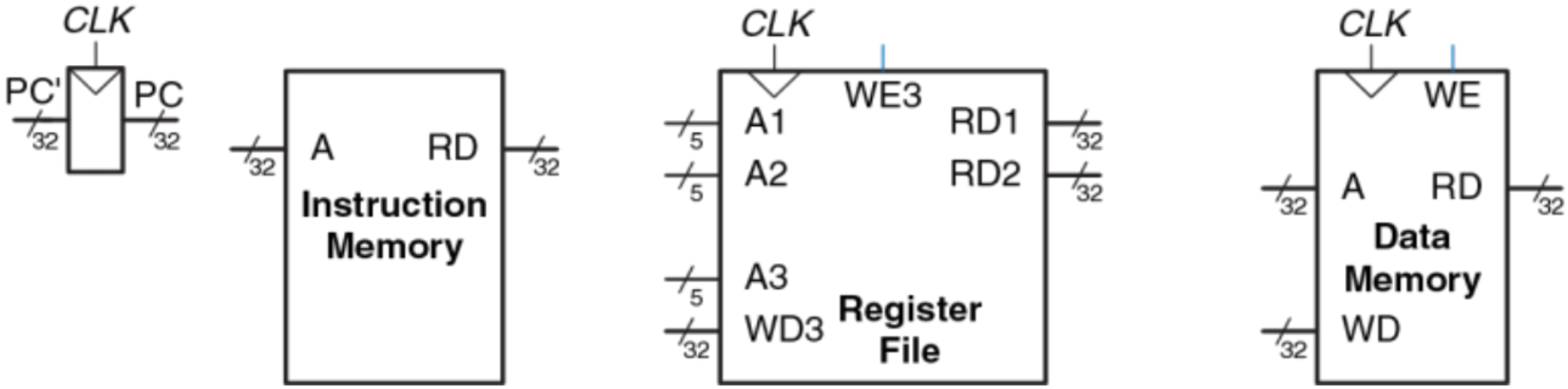
У некоторых микроархитектур есть также и *неархитектурное* (т.е. невидимое программисту) состояние, которое используется или для упрощения логики, или для улучшения производительности.

Для простоты понимания, рассмотрим небольшое подмножество набора команд MIPS, именно:
– Арифметические и логические команды типа R: add, sub, and, or, slt
– Команды доступа в память: lw, sw
– Команды условного перехода: beq

Две взаимодействующих части: тракт данных и устройство управления.
Тракт данных содержит: память, регистры, АЛУ и мультиплексоры (32-бита).
Устройство управления получает текущую команду из тракта данных и в ответ говорит ему, как именно выполнять эту команду. В частности, УУ генерирует адресные сигналы для мультиплексоров, сигналы разрешения работы для регистров и сигналы разрешения записи в память.

Начнем с элементов, хранящих состояние системы: память для хранения команд и данных, блоки для хранения архитектурного состояния.
Затем между этими элементами нужно будет расположить комбинационные схемы, вычисляющие новое состояние на основе текущего состояния.

Разделим память на две части: одна содержит команды, другая – данные.



Элементы, хранящие состояние процессора

Толщиной линии обозначена ширина шины.
Синия линия – сигнал разрешения записи.
У элементов, хранящих состояние системы, обычно есть сигнал сброса, который устанавливает их в известное состояние в момент включения (не показан).

Счетчик команд – обычный 32-битный регистр.
Выход (PC) содержит адрес текущей команды.
Вход (PC′) содержит адрес следующей команды.

Из всех регистров, как минимум (PC) должен иметь сигнал сброса, который проинициализирует его в момент включения процессора.
При получении сигнала сброса PC инициализируется значением 0xBFC00000.
Как только сигнал сброса снят, процессор начинает выполнять код по этому адресу. Этот код загружает операционную систему (ОС). Затем ОС загружает в память по адресу 0x00400000 пользовательскую программу и передает ей управление.
Для простоты считаем, что PC инициализируется значением 0x00000000.

Память команд имеет один порт чтения.
На адресный вход A подается 32-битный адрес команды, после чего на выходе RD появляется 32-битное число (команда), прочитанное из памяти по этому адресу.

Регистровый файл (32 элемента по 32 бита), имеет два порта чтения и один порт записи данных.
Порты чтения имеют пятибитные входы адреса A1 и A2.
Каждый из портов читает 32-битное значение из регистра и подает его на выходы RD1 и RD2 соответственно.
Порт записи получает пятибитный адрес регистра на адресный вход A3, 32-битное число на вход данных WD, сигнал разрешения записи WE3 и тактовый сигнал.
Если сигнал разрешения записи равен единице, то регистровый файл записывает данные в указанный регистр по положительному фронту тактового сигнала.

Память данных имеет один порт чтения/записи. Если сигнал разрешения записи WE равен единице, то данные со входа WD записываются в ячейку памяти с адресом A по положительному фронту тактового сигнала. Если же сигнал разрешения записи равен нулю, то данные из ячейки с адресом A подаются на выход RD.

Чтение из памяти команд, регистрового файла и памяти данных происходит асинхронно, то есть независимо от тактового сигнала.
Запись же производится исключительно по положительному фронту тактового Сигнала – состояние системы изменяется только по фронту тактового сигнала.

Можно рассмотреть три типа микроархитектуры:

– однотактную,
– многотактную,
– конвейерную.

Они различаются способом соединения элементов памяти, а также наличием или отсутствием неархитектурного состояния.

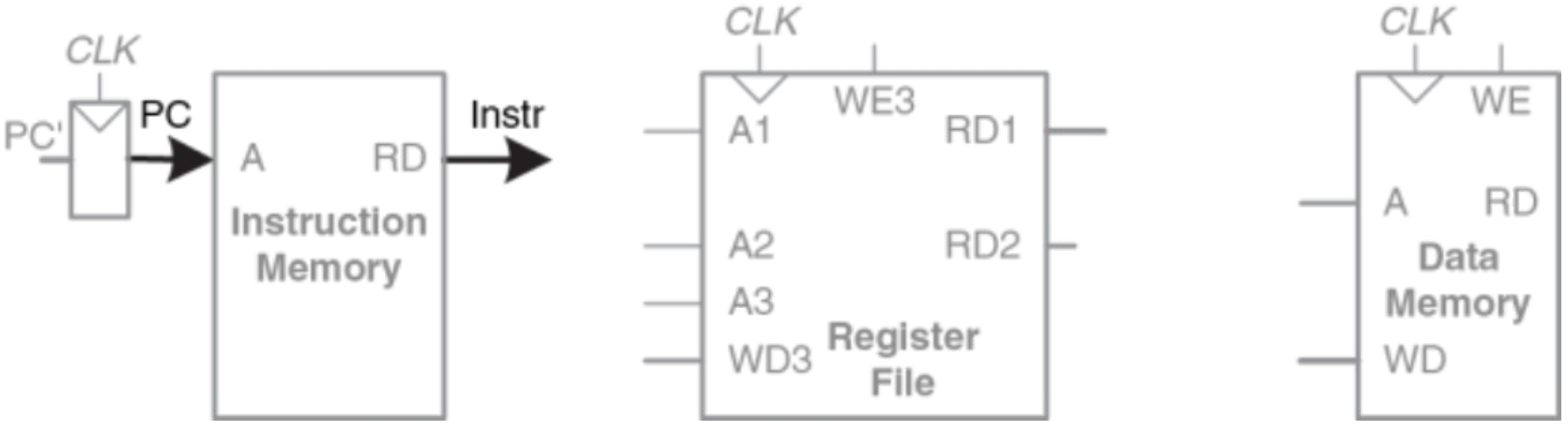
Однотактная микроархитектура выполняет всю команду за один такт.
Самое простое УУ.
Все действия выполняются за один такт → не требуется неархитектурного состояния. Длительность такта ограничена самой медленной командой.

Многотактная микроархитектура выполняет команду за несколько более коротких тактов. Простым командам нужно меньше тактов, чем сложным.
ММ уменьшает количество аппаратуры путем повторного использования таких «дорогих» блоков, как сумматоры и блоки памяти. Например, при выполнении команды один и тот же сумматор на разных тактах может быть использован для разных целей.
Повторное использование блоков достигается путем добавления в многотактный процессор нескольких неархитектурных регистров для сохранения промежуточных результатов.
Многотактный процессор выполняет только одну команду за раз, и каждая команда занимает несколько тактов.

Конвейерная микроархитектура – результат применения принципа конвейерной обработки к однотактной микроархитектуре.
Она позволяет выполнять несколько команд одновременно, значительно улучшая пропускную способность процессора.
КМ требует дополнительной логики для разрешения конфликтов между одновременно выполняемыми в конвейере командами. Она также требует несколько неархитектурных регистров, расположенных между стадиями конвейера.
(Все коммерческие высокопроизводительные процессоры используют конвейеры.)

Сконструируем тракт данных путем соединения, хранящих состояние процессора, при помощи комбинационной логики, которая и будет выполнять разные команды. Управляющие сигналы нужны, чтобы указывать, как именно тракт данных должен выполнять команду, находящуюся в нем в текущий момент времени.
УУ содержит комбинационную логику, которая формирует управляющие сигналы в зависимости от того, какая команда выполняется в данный момент.

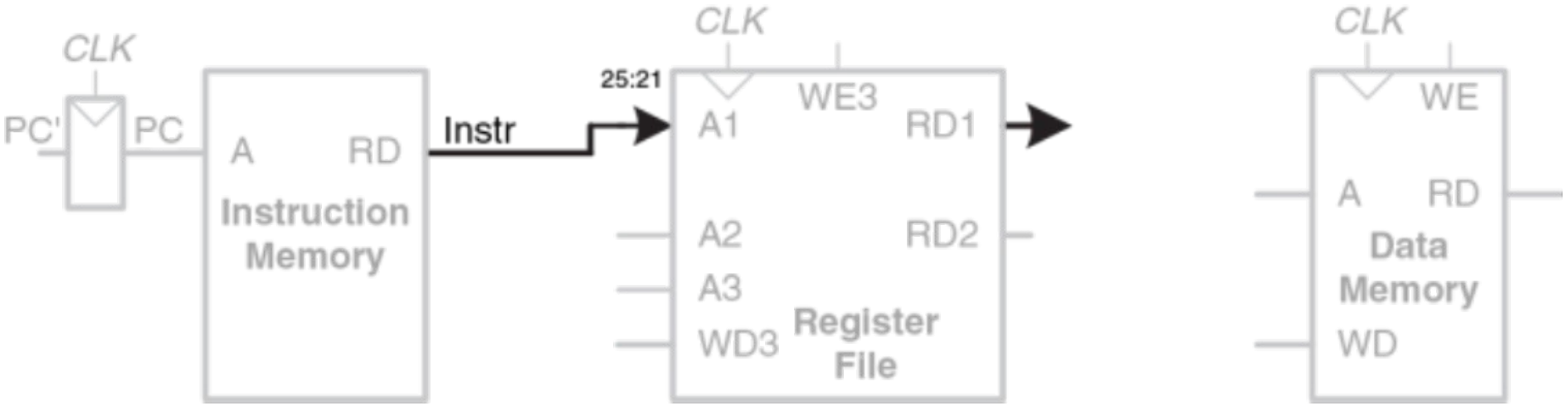
Первый шаг: чтение команды из памяти команд (выборка).



Дальнейшие действия процессора будут зависеть от того, какая именно команда была выбрана.
Создадим тракт данных для команды lw, после чего расширим его для других команд.

Однотактный тракт данных

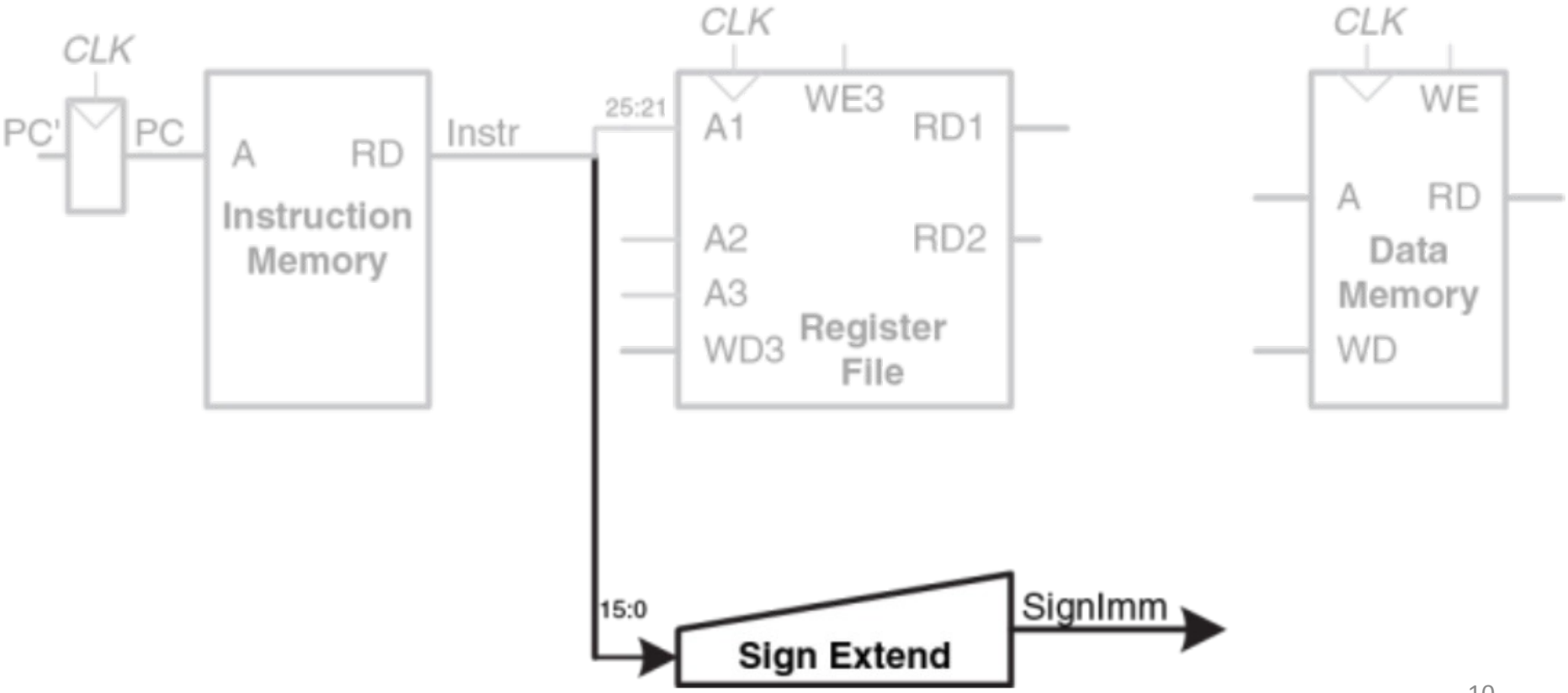
Для команды lw следующим шагом мы должны прочитать регистр операнда (source register), содержащий базовый адрес.
Номер этого регистра указан в поле rs (Instr[25:21]).
Эти пять битов подключены к адресному входу первого порта (A1) регистрового файла.
Значение, прочитанное из регистрового файла, появляется на его выходе RD1.



Команде lw также требуется смещение (offset) – число, которое будет прибавлено к базовому адресу. Смещение передается как непосредственный операнд (immediate), то есть находится непосредственно в поле Instr[15:0], занимая младшие 16 бит.

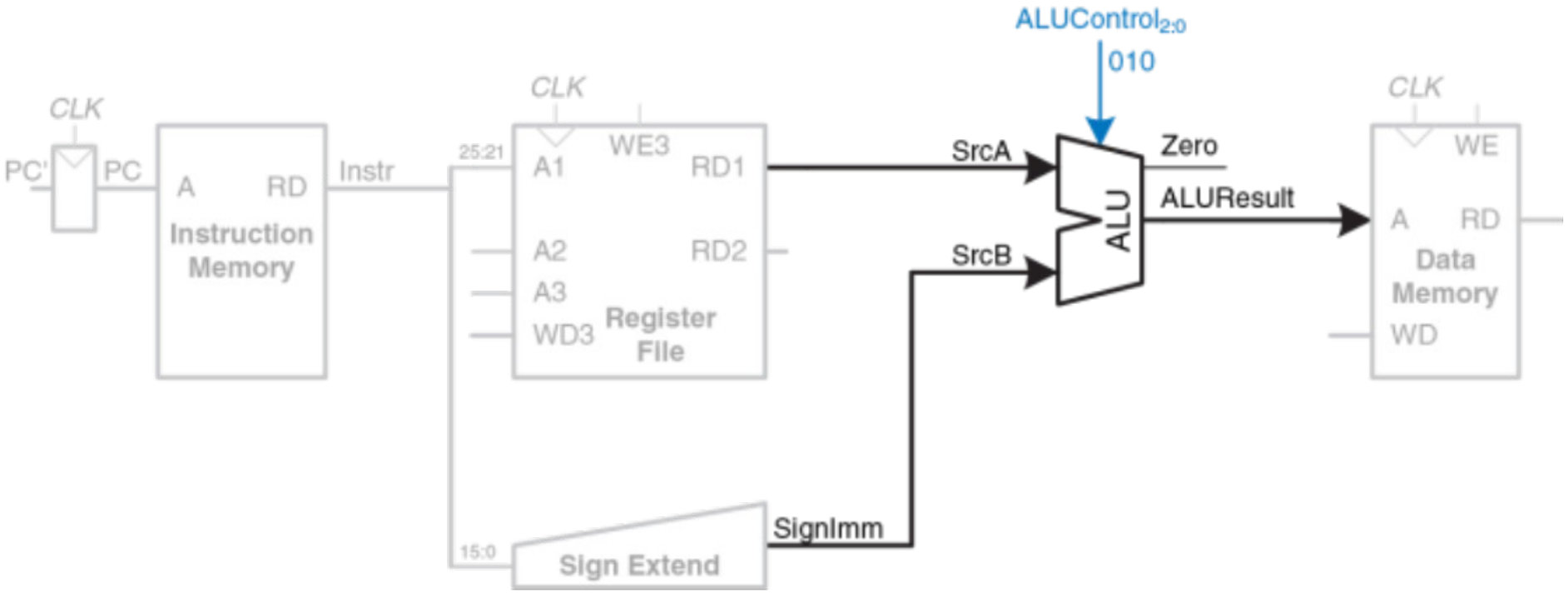
16-битное число может быть как положительным, так и отрицательным, и над ним должна быть выполнена операция знакового расширения до 32 бит.

SignImm[15:0] = Instr[15:0], SignImm[31:16] = Instr[15].



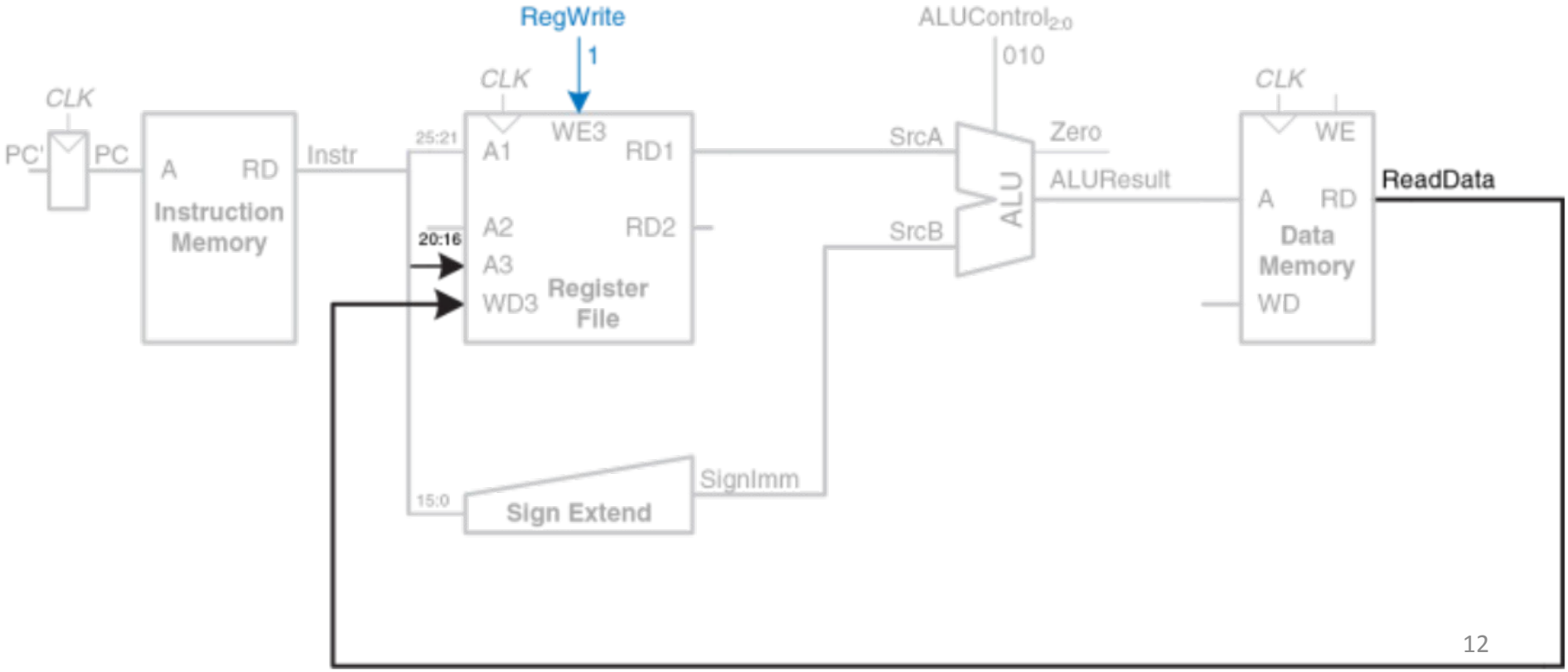
Однотактный тракт данных

Процессор должен добавить смещение к базовому адресу, чтобы получить адрес, по которому будет произведено чтение из памяти.
Для суммирования в тракт данных добавляется АЛУ (ALU).
Трехбитный управляющий сигнал ALUControl говорит АЛУ, какую операцию надо выполнить. На выходе из АЛУ получается 32-битный результат ALUResult и флаг нуля (Zero flag), который устанавливается в единицу, если ALUResult равен нулю.
Далее ALUResult подается на адресный вход памяти данных



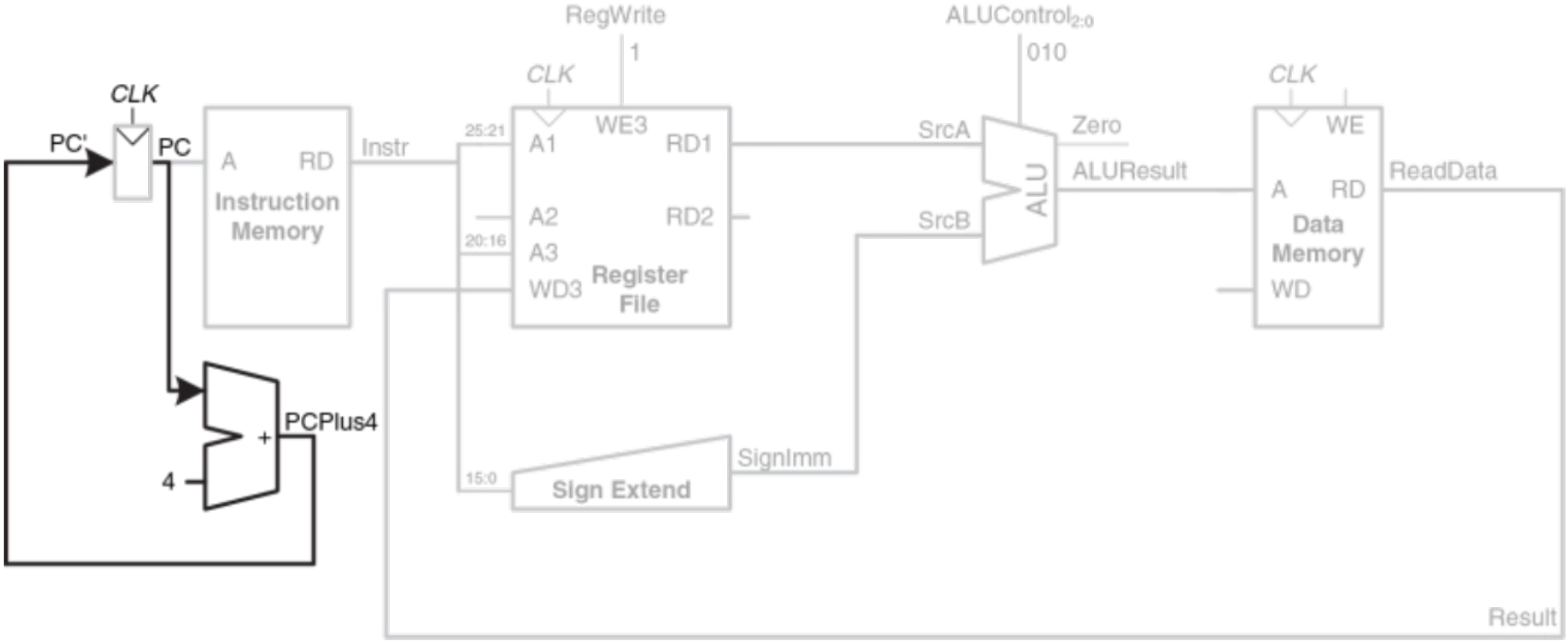
Однотактный тракт данных

Значение, прочитанное из памяти данных, попадает на шину ReadData, после чего записывается обратно в регистровый файл в конце такта.
Регистр результата (destination register), в который команда lw запишет прочитанное из памяти значение, определяется полем rt (Instr[20:16]), которое подключено к адресному входу третьего порта (A3) регистрового файла. Управляющий сигнал RegWrite соединен со входом разрешения записи третьего порта (WE3) и активен во время выполнения команды lw, чтобы прочитанное значение было записано в регистровый файл.
Сама запись происходит по положительному фронту тактового сигнала, которым заканчивается такт процессора.



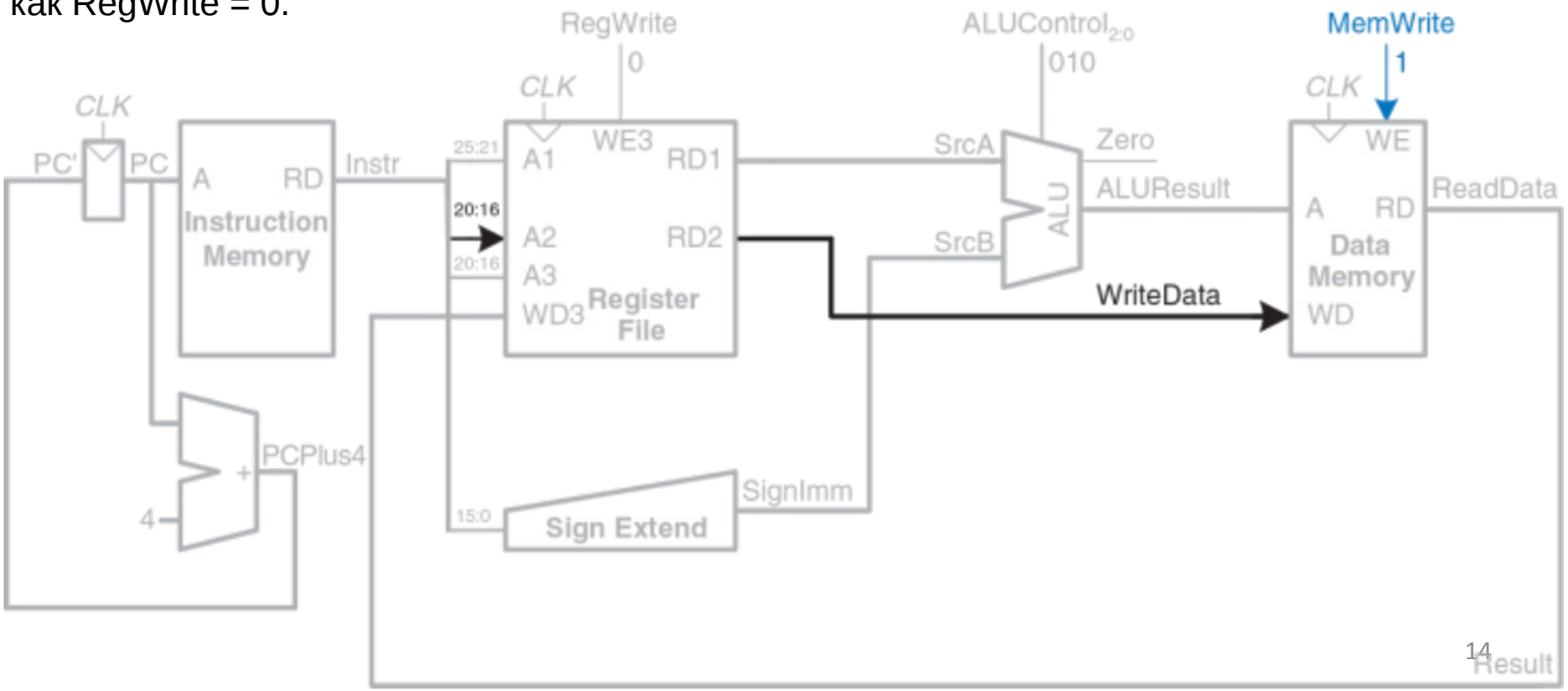
Однотактный тракт данных

Одновременно с выполнением команды процессор должен вычислить адрес следующей команды, PC′. Так как команды 32-битные (4 байта), то адрес следующей команды равен PC + 4 – нужен еще один сумматор.
Новый адрес записывается в счетчик команд в момент прихода положительного фронта тактового сигнала.



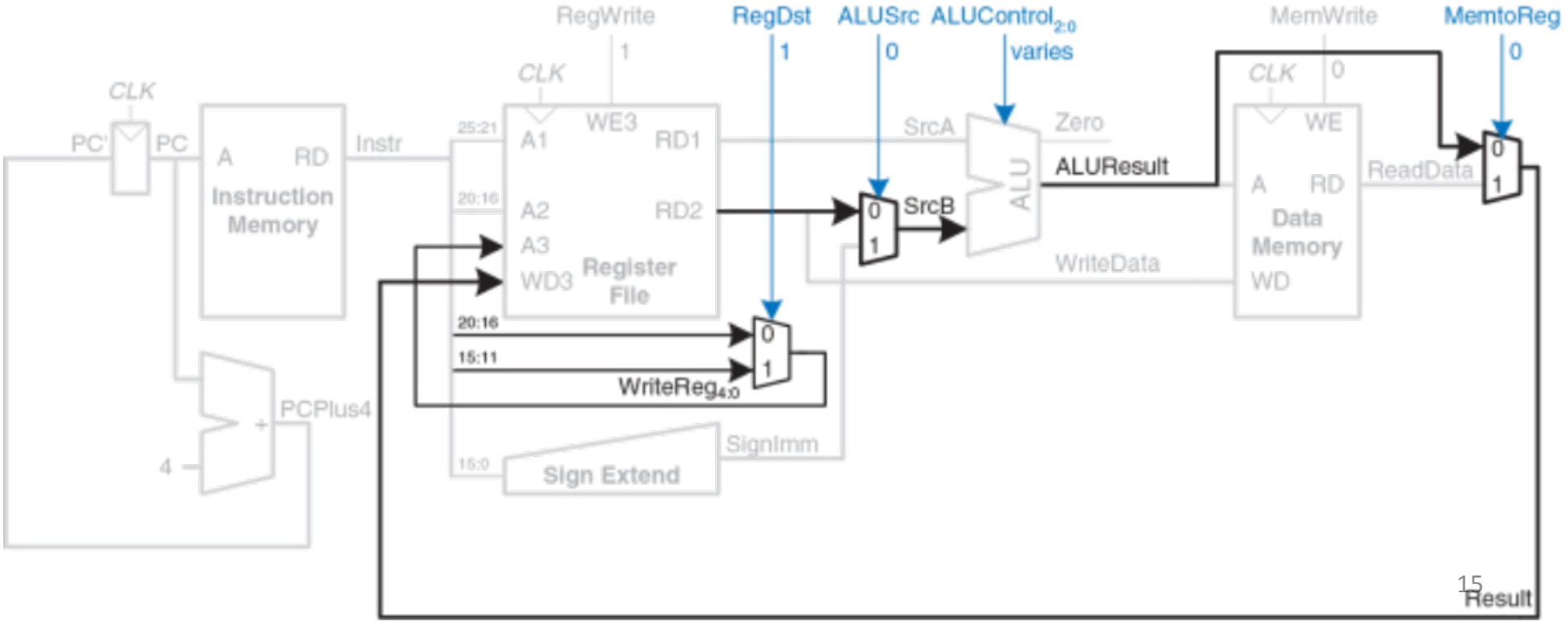
Однотактный тракт данных

Улучшим тракт данных, чтобы он мог выполнять еще и команду sw.
Команда sw читает из регистрового файла еще один регистр и записывает его содержимое в память данных. Номер регистра указывается в поле rt (Instr[20:16]). Эти пять бит подключены ко второму порту (A2) регистрового файла.
Вход разрешения записи (WE) управляется сигналом MemWrite. Для команды sw сигнал MemWrite = 1, чтобы данные были записаны в память; ALUControl = 010, чтобы базовый адрес был просуммирован со смещением; и RegWrite = 0, потому что команда ничего не пишет в регистровый файл.
ReadData читается из памяти в любом случае, но прочитанное значение игнорируется, так как RegWrite = 0.



Однотактный тракт данных

Теперь добавим поддержку команд типа R – add, sub, and, or, и slt.
Эти команды читают два регистра из регистрового файла, выполняют над ними некие операции в АЛУ и записывают результат обратно в третий регистр.
Единственное различие этих команд – в типе операции. Все они могут быть выполнены одной и той же аппаратурой, используя разные значения управляющего сигнала ALUControl.
Содержимое двух регистров читается из регистрового файла и подается на входы АЛУ.
Команды типа R должны записывать в регистровый файл значение ALUResult. Чтобы выбирать между ReadData и ALUResult, добавлен еще один мультиплексор.
Для команд типа R номер регистра задается полем rd (Instr[15:11]), и добавлен третий мультиплексор, чтобы присваивать сигналу WriteReg значение из нужного поля.
Этот мультиплексор управляется сигналом RegDst.

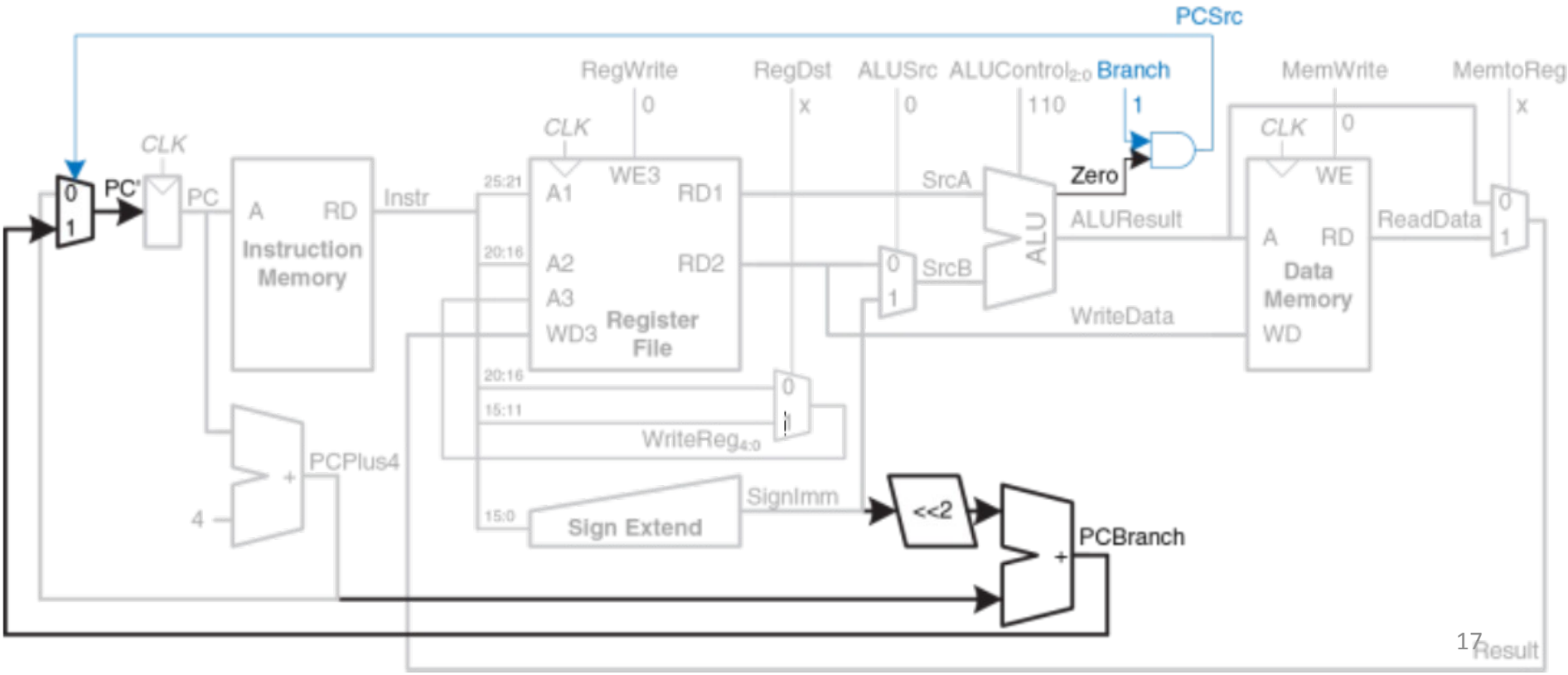


Однотактный тракт данных

Добавим поддержку поддержку команды beq.
Эта команда сравнивает два регистра, и, если они равны, то добавляет смещение к счетчику команд PC, выполняя условный переход.
Смещение – это положительное или отрицательное число, передаваемое как непосредственный операнд в поле Instr[15:0].
Смещение указывает, сколько команд нужно перепрыгнуть, прежде чем продолжить выполнение программы. Т.о., над значением непосредственного операнда надо выполнить операцию знакового расширения, после чего умножить его на четыре:
$PC' = PC + 4 + SignImm \times 4$.

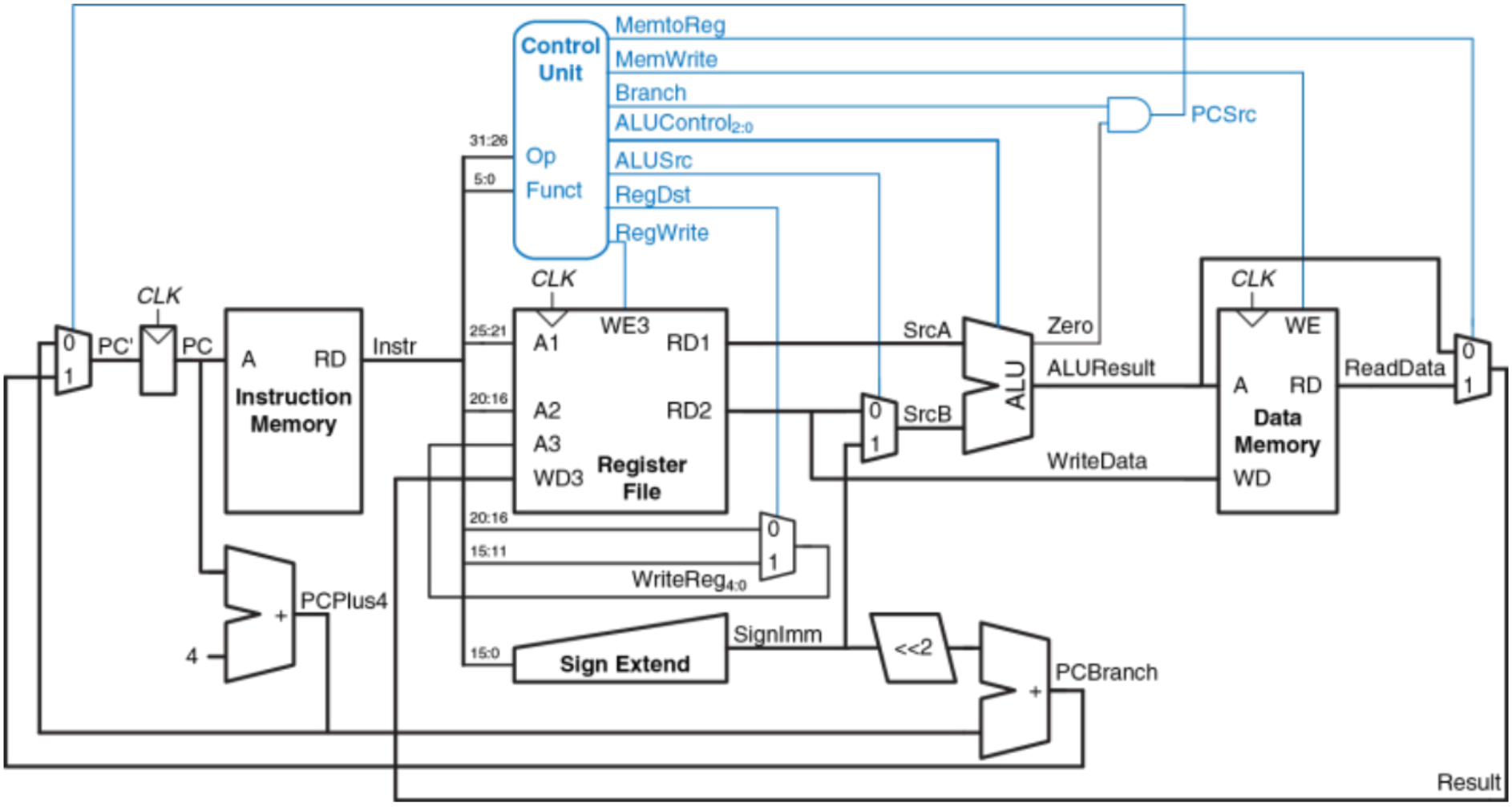
Новое значение счетчика команд на случай выполненного условного перехода (PCBranch) вычисляется путем сдвига SignImm влево на два разряда и последующего сложения с PCPlus4.
Сдвиг влево на два разряда – это легкий способ умножения на четыре, так как сдвиг на константное число разрядов не требует никаких логических элементов, а требует только переподключения сигналов.
Два регистра сравниваются путем вычитания одного из другого в АЛУ.
Если ALUResult равен нулю, о чем сигнализирует флаг нуля, то регистры равны.
Нужно добавить мультиплексор, чтобы выбрать, какое именно значение присвоить PC′: PCPlus4 или PCBranch.

PCBranch используется тогда, когда выполняется команда условного перехода и установлен флаг нуля.
T.o., сигнал Branch равен единице для команды beq и нулю для прочих команд.
Для beq сигналы ALUControl = 110 (вычитание).
ALUSrc = 0, чтобы операнд SrcB был прочитан из регистрового файла.
RegWrite и MemWrite равны нулю, так как команда условного перехода ничего не пишет.



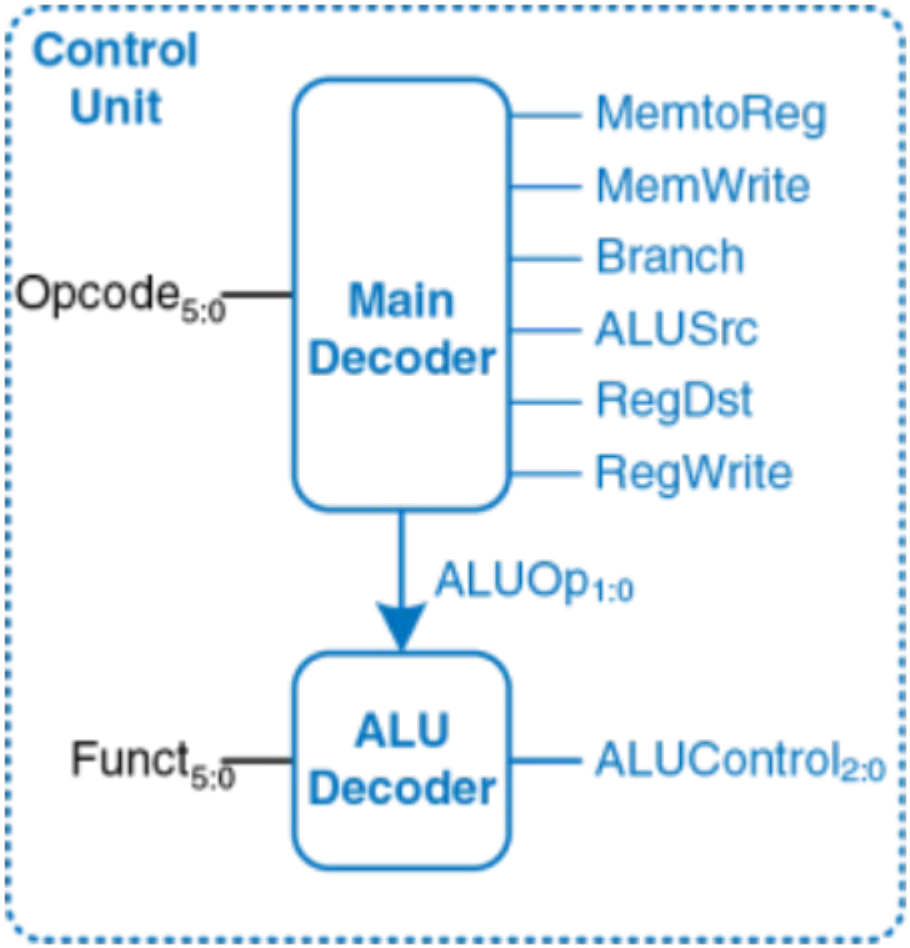
Однотактное устройство управления

Устройство управления формирует управляющие сигналы на основе полей opcode и funct, присутствующих в каждой команде (Instr[31:26] и Instr[5:0]).



Однотактное устройство управления

Большая часть информации для УУ берется из поля opcode, но команды типа R также используют и поле funct для определения операции в АЛУ.
Для упрощения разработки, поделим УУ на две части.



Основной дешифратор вычисляет значение большинства выходов на основе поля opcode.
Он также формирует двухбитный сигнал ALUOp.

Дешифратор АЛУ (ALU decoder) использует ALUOp совместно с полем funct для вычисления состояния ALUControl.

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

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)

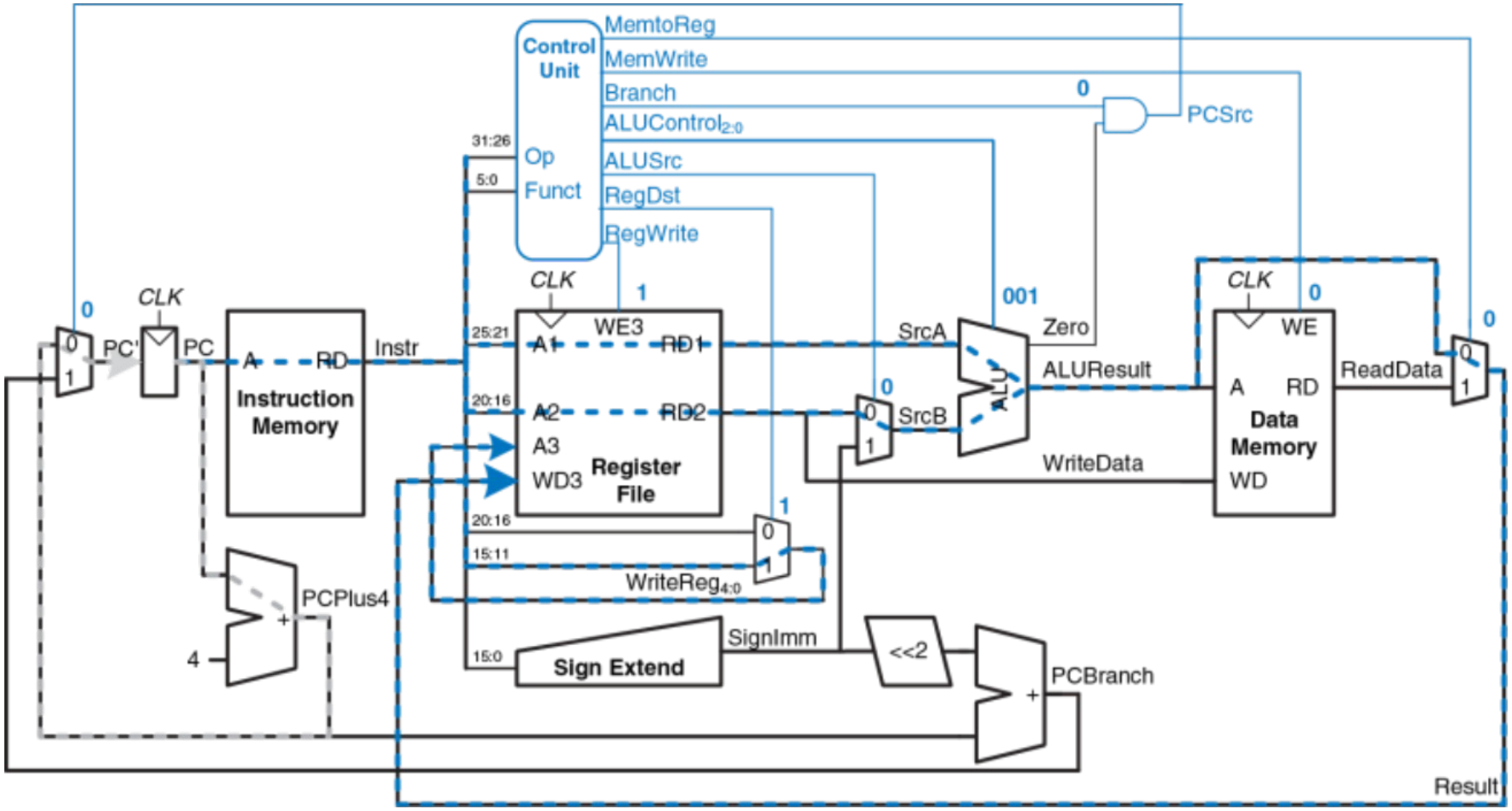
Таблица истинности основного дешифратора

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

Для всех команд типа R основной дешифратор формирует одинаковые сигналы; эти команды отличаются только сигналами, сформированными дешифратором АЛУ.
Для команд, которые не пишут в регистровый файл (например, sw или beq), управляющие сигналы RegDst и MemtoReg могут принимать любое состояние, то есть являются неопределенными (X); адрес и данные, приходящие на порт записи регистрового файла, не имеют никакого значения, так как RegWrite равен нулю.

Пример

Управляющие сигналы и пути движения данных для команды or.



Введение дополнительных команд

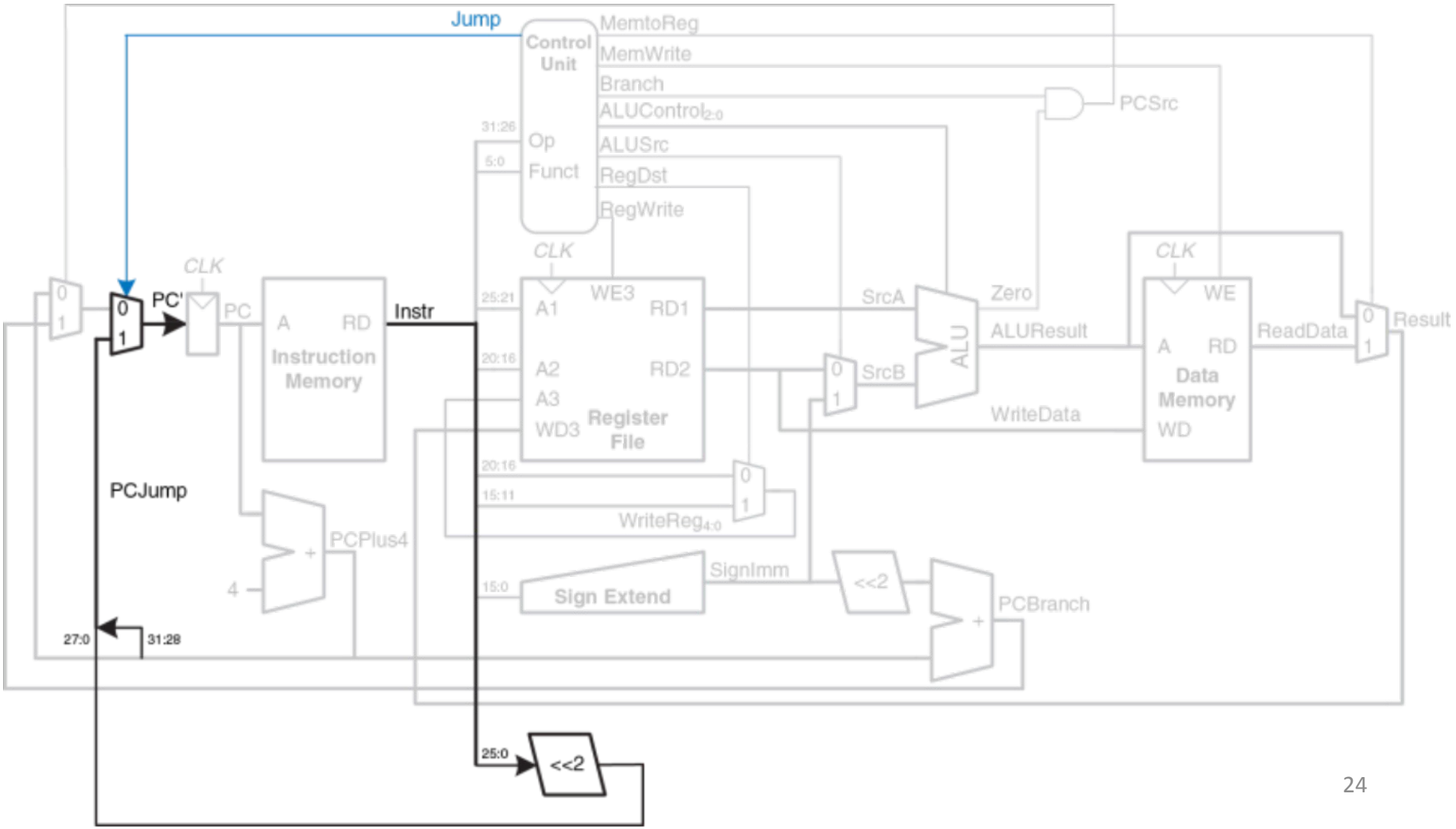
Команда сложения с непосредственным операндом (addi) складывает значение одного из регистров со значением, хранящимся непосредственно в поле команды, и записывает результат в другой регистр.

Что нужно сделать: добавить новую строку в таблицу истинности основного дешифратора и заполнить ее значениями управляющих сигналов для команды 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

Введение дополнительных команд

Команда безусловного перехода j записывает новое значение в счетчик команд. Два младших бита нового значения всегда равны нулю. Следующие 26 бит процессор возьмет из поля адреса перехода (jump address field) Instr[25:0]. Четыре оставшихся старших бита будут равны четырем старшим битам предыдущего значения PC.

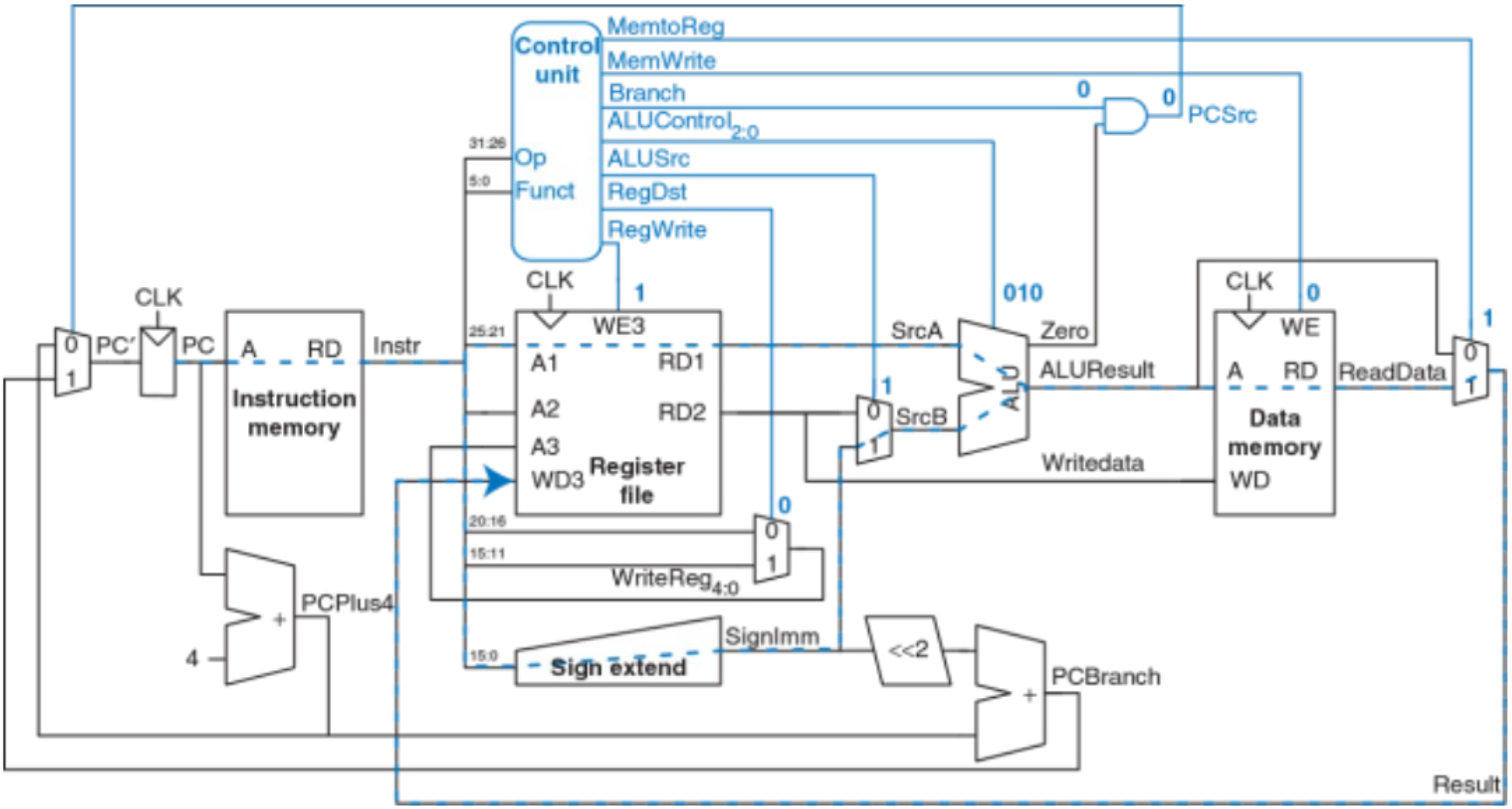


Введение дополнительных команд

Команда безусловного перехода j – дополнение основного дешифратора.

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

Цепь с наибольшей задержкой для команды lw



У других команд задержка может быть меньше.

Задержки элементов (65-нм КМОП-техпроцесс).

Элемент	Параметр	Задержка (пс)
Задержка распространения clk-to-Q в регистре	t_{pcq}	30
Время предустановки регистра	t_{setup}	20
Мультиплексор	t_{mux}	25
АЛУ	t_{ALU}	200
Чтение из памяти	t_{mem}	250
Чтение из регистрового файла	t_{RFread}	150
Время предустановки регистрового файла (register file setup)	tRF_{setup}	20

Недостатки однотактного процессора

У однотактного процессора три основных проблемы:

– период его тактового сигнала должен быть достаточно большим, чтобы успела выполниться самая медленная команда (lw), несмотря на то, что большинство остальных команд гораздо быстрее.

– нужно три сумматора (один для АЛУ и два для вычисления нового значения счетчика команд); сумматоры, особенно быстрые, требуют множества логических элементов, что делает их относительно дорогими схемами.

– требуется отдельная память команд и данных (гарвардская архитектура). В большинстве компьютеров используют общую память для команд и данных, доступную для чтения и записи.