Глава 7

Цифровая схемотехника и архитектура компьютера, второе издание

Дэвид М. Харрис и Сара Л. Харрис

МИКРОАРХИТЕКТУРА

ELSEVIER

Цифровая схемотехника и архитектура компьютера

Эти слайды предназначены для преподавателей, которые читают лекции на основе учебника «Цифровая схемотехника и архитектура компьютера» авторов Дэвида Харриса и Сары Харрис. Бесплатный русский перевод второго издания этого учебника можно загрузить с сайта компании Imagination Technologies:

https://community.imgtec.com/downloads/digital-design-and-computer-architecture-russian-edition-second-edition

Процедура регистрации на сайте компании Imagination Technologies описана на станице:

http://www.silicon-russia.com/2016/08/04/harris-and-harris-2/

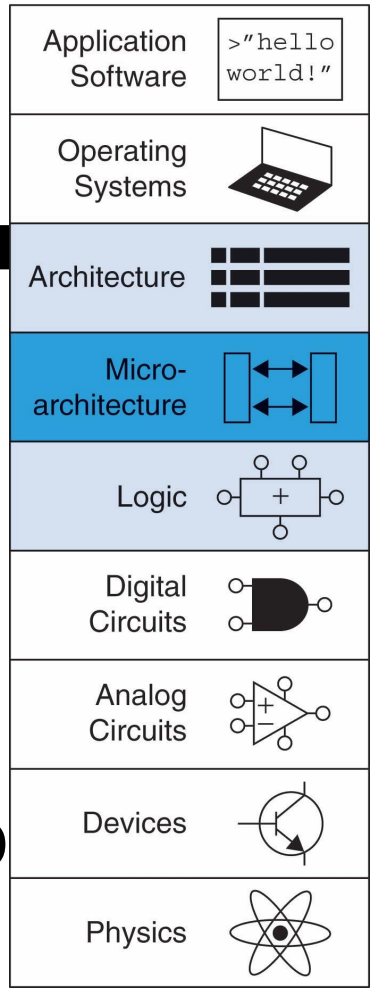
Благодарности

Перевод данных слайдов на русский язык был выполнен командой сотрудников университетов и компаний из России, Украины, США в составе:

- Александр Барабанов - доцент кафедры компьютерной инженерии факультета радиофизики, электроники и компьютерных систем Киевского национального университета имени Тараса Шевченко, кандидат физ.-мат. наук, Киев, Украина;
- Антон Брюзгин - начальник отдела АО «Вибро-прибор», Санкт-Петербург, Россия.
- Евгений Короткий - доцент кафедры конструирования электронно-вычислительной аппаратуры факультета электроники Национального технического университета Украины «Киевский Политехнический Институт», руководитель открытой лаборатории электроники Lampa, кандидат технических наук, Киев, Украина;
- Евгения Литвинова – заместитель декана факультета компьютерной инженерии и управления, доктор технических наук, профессор кафедры автоматизации проектирования вычислительной техники Харьковского национального университета радиоэлектроники, Харьков, Украина;
- Юрий Панчул - старший инженер по разработке и верификации блоков микропроцессорного ядра в команде MIPS I6400, Imagination Technologies, отделение в Санта-Кларе, Калифорния, США;
- Дмитрий Рожко - инженер-программист АО «Вибро-прибор», магистр Санкт-Петербургского государственного автономного университета аэрокосмического приборостроения (ГУАП), Санкт-Петербург, Россия;
- Владимир Хаханов – декан факультета компьютерной инженерии и управления, проректор по научной работе, доктор технических наук, профессор кафедры автоматизации проектирования вычислительной техники Харьковского национального университета радиоэлектроники, Харьков, Украина;
- Светлана Чумаченко – заведующая кафедрой автоматизации проектирования вычислительной техники Харьковского национального университета радиоэлектроники, доктор технических наук, профессор, Харьков, Украина.

Глава 7 :: Темы

- Введение
- Анализ производительности
- Однотактный процессор
- Многотактный процессор
- Конвейерный процессор
- Исключения
- Улучшение микроархитектур



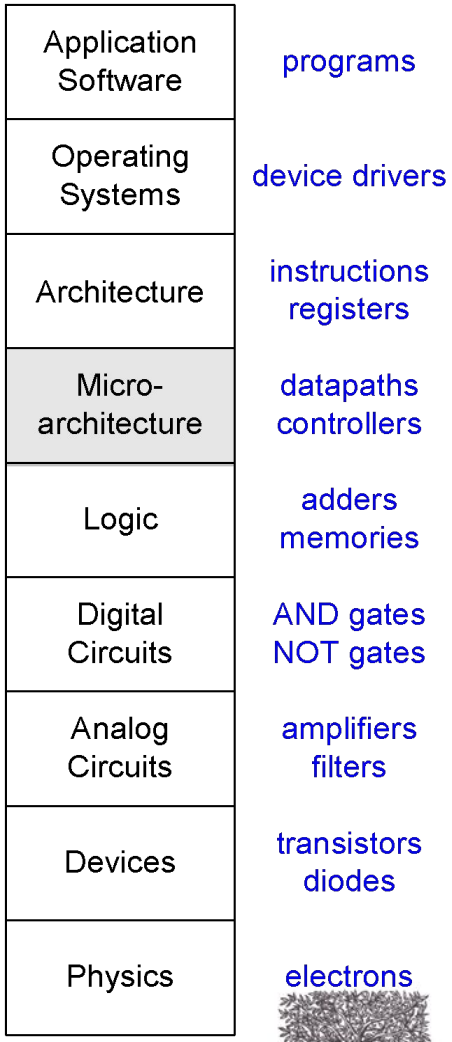
МИКРОАРХИТЕКТУРА

ELSEVIER

МИКРОАРХИТЕКТУРА

Введение

- **Микроархитектура:** аппаратная реализация архитектуры в виде схемы
- Процессор:
  - **Тракт данных:** функциональные блоки обработки и передачи данных (арифметико-логическое устройство, регистровый файл, мультиплексоры и т.д.)
  - **Устройство управления:** формирует управляющие сигналы для функциональных блоков



ELSEVIER

МИКРОАРХИТЕКТУРА

Микроархитектура

- Несколько аппаратных реализаций одной и той же архитектуры:
  - **Однотактная реализация:** каждая инструкция выполняется за один такт
  - **Многотактная реализация:** каждая инструкция разбивается на несколько шагов и выполняется за несколько тактов
  - **Конвейерная реализация:** каждая инструкция разбивается на несколько шагов и несколько инструкций выполняются одновременно

Производительность

- Время выполнения программы

Execution Time = (#instructions)(cycles/instruction)(seconds/cycle)

Время выполнения = (#инструкции)(такты/инструкция)(секунды/такт)

- Определения:
  - CPI: Количество тактов на выполнение инструкции (Cycles/instruction)
  - Период тактовой частоты: секунды/такт
  - IPC: Количество инструкций выполняемых за такт (instructions/cycle = IPC = 1 / CPI)
- Необходимо удовлетворять следующие ограничения:
  - Стоимость
  - Площадь на кристалле
  - Энергопотребление
  - Производительность

MIPS процессор

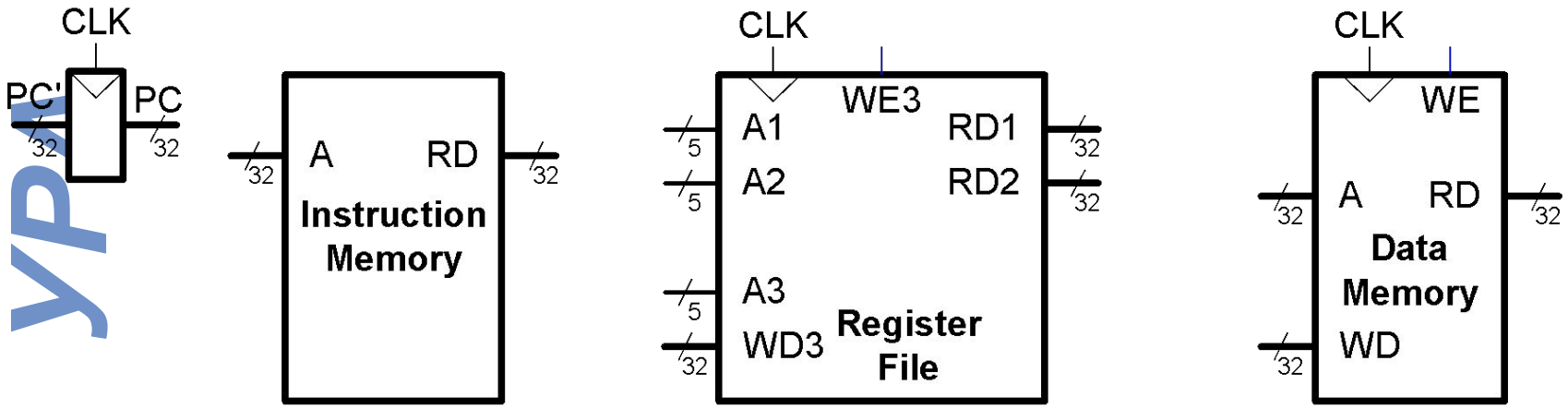
- Будем рассматривать подмножество инструкций MIPS:
  - Инструкции R-типа: `and`, `or`, `add`, `sub`, `slt`
  - Инструкции работы с памятью: `lw`, `sw`
  - Инструкции переходов: `beq`, `j`

МИКРОАРХИТЕКТУРА

Архитектурное состояние

- Определяется:
  - Содержимым счетчика команд (PC)
  - Содержимым 32-х регистров общего назначения
  - Содержимым памяти

Элементы, хранящие состояние MIPS



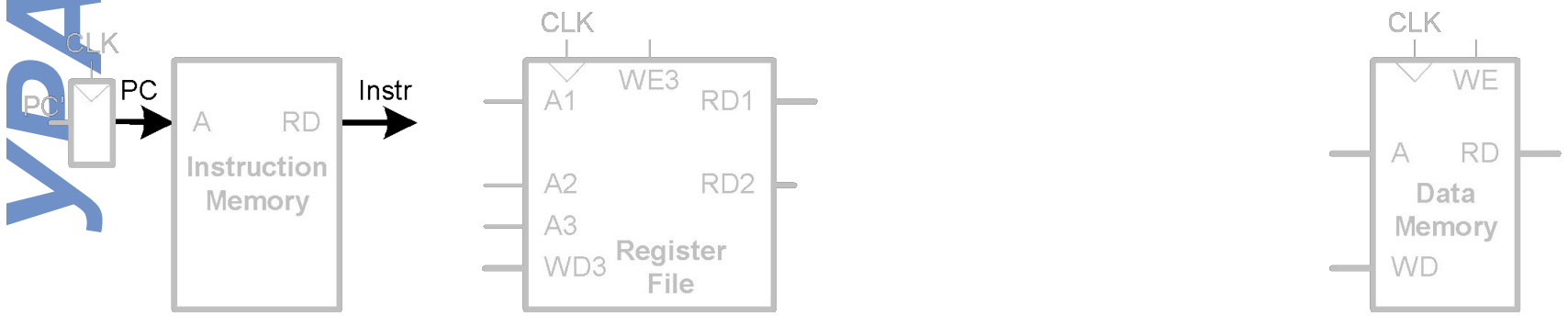
МИКРОАРХИТЕКТУРА

Однотактный MIPS процессор

- Тракт данных
- Устройство управления

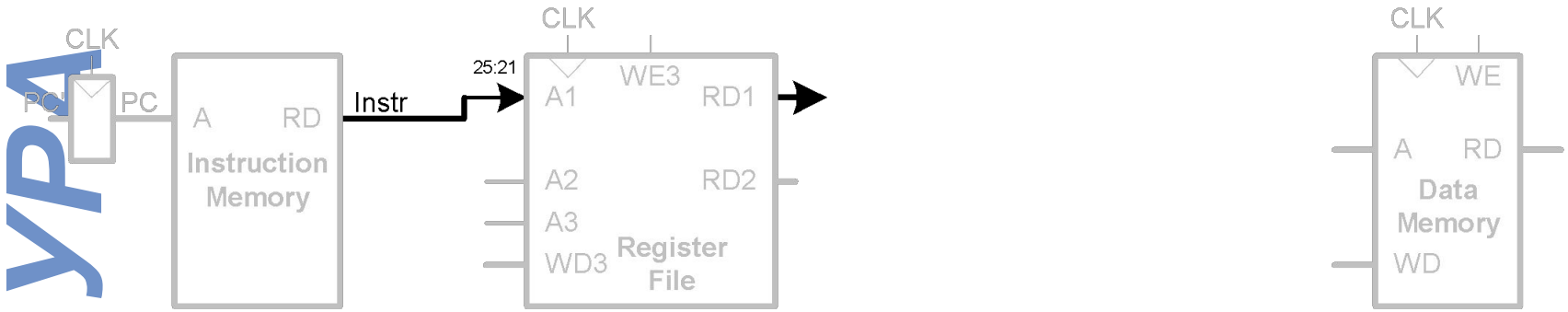
Однотактный тракт данных: выборка lw

Шаг 1: Выборка (считывание) инструкции `lw` из памяти



Однотактный тракт данных: чтение регистров

Шаг 2: считывание операндов-источников из регистрового файла



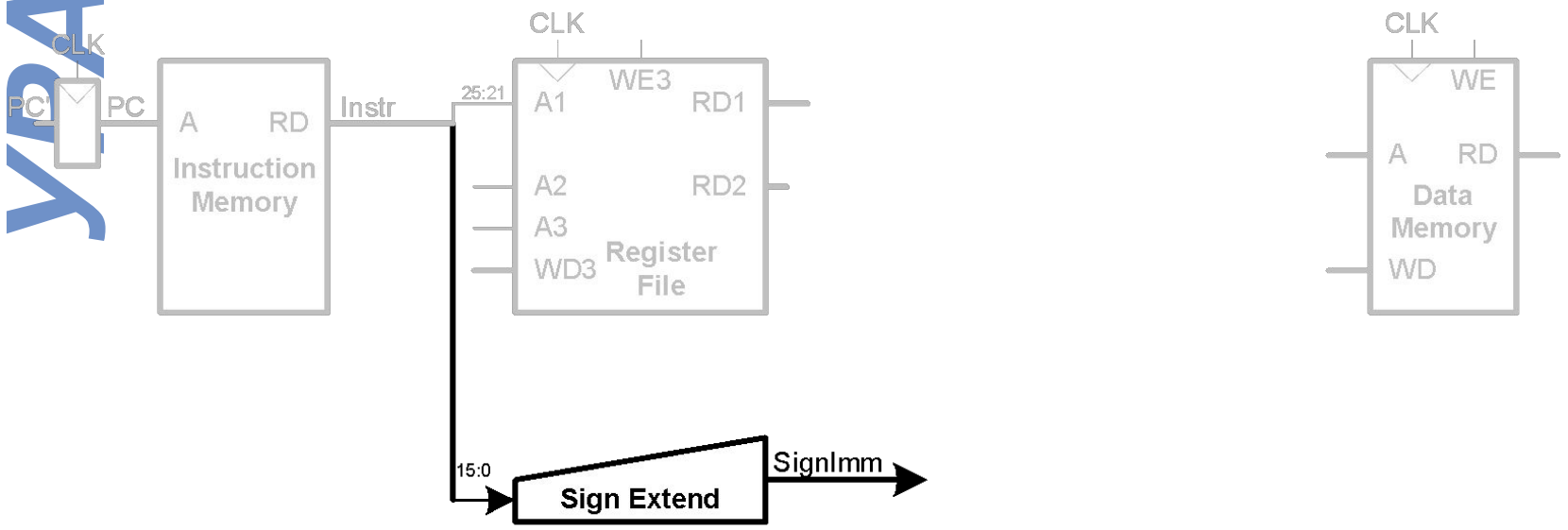

```
lw    rt, imm(rs)
```

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

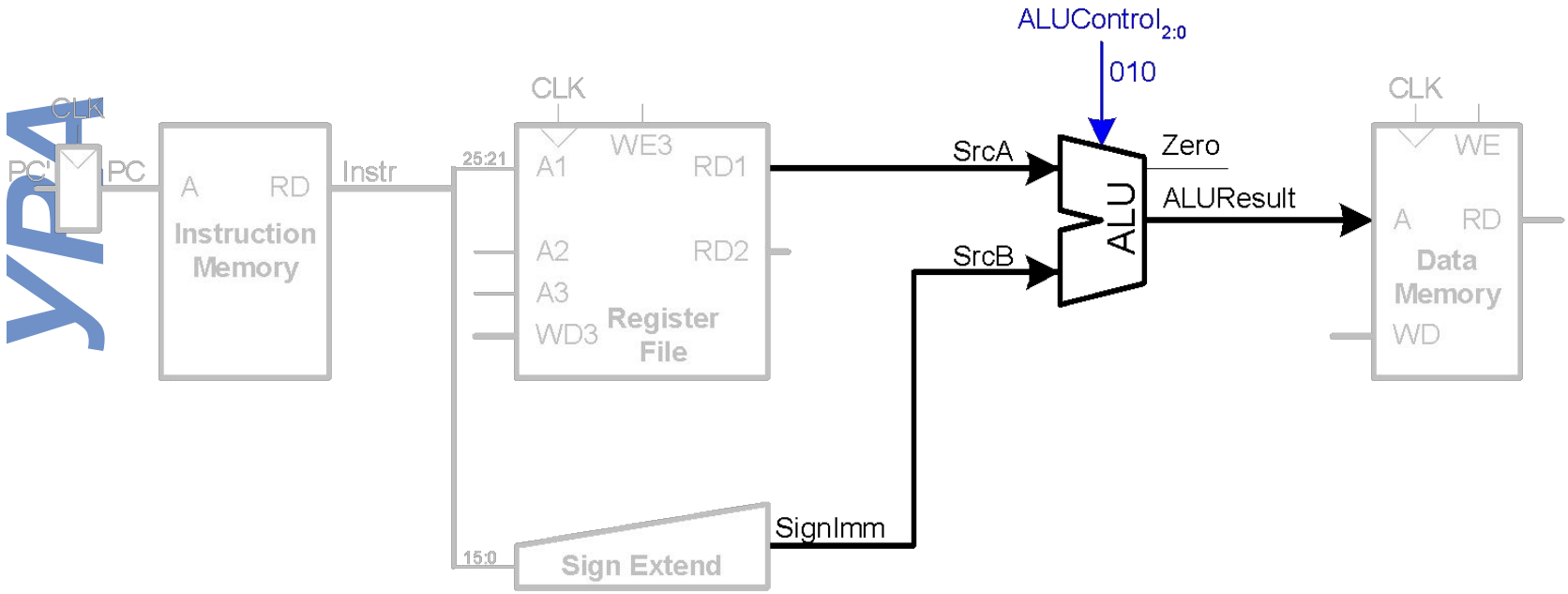
Однотактный тракт данных: расширение константы

Шаг 3: расширение 16-битной константы до 32-х разрядов битом знака



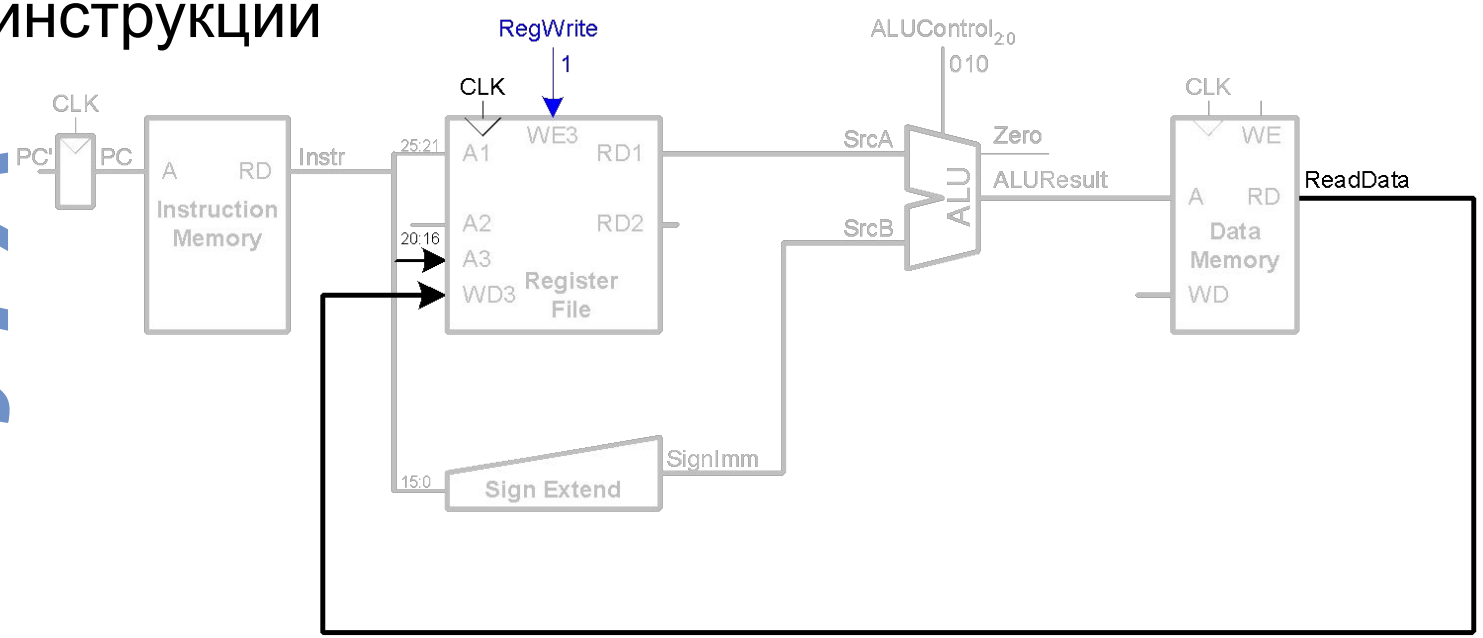
Однотактный тракт данных: вычисление адреса

Шаг 4: Вычисление адреса ячейки в памяти



Однотактный тракт данных: считывание из памяти

- **Шаг 5:** считываем данные из памяти и записываем их в регистр, номер которого хранится в коде инструкции



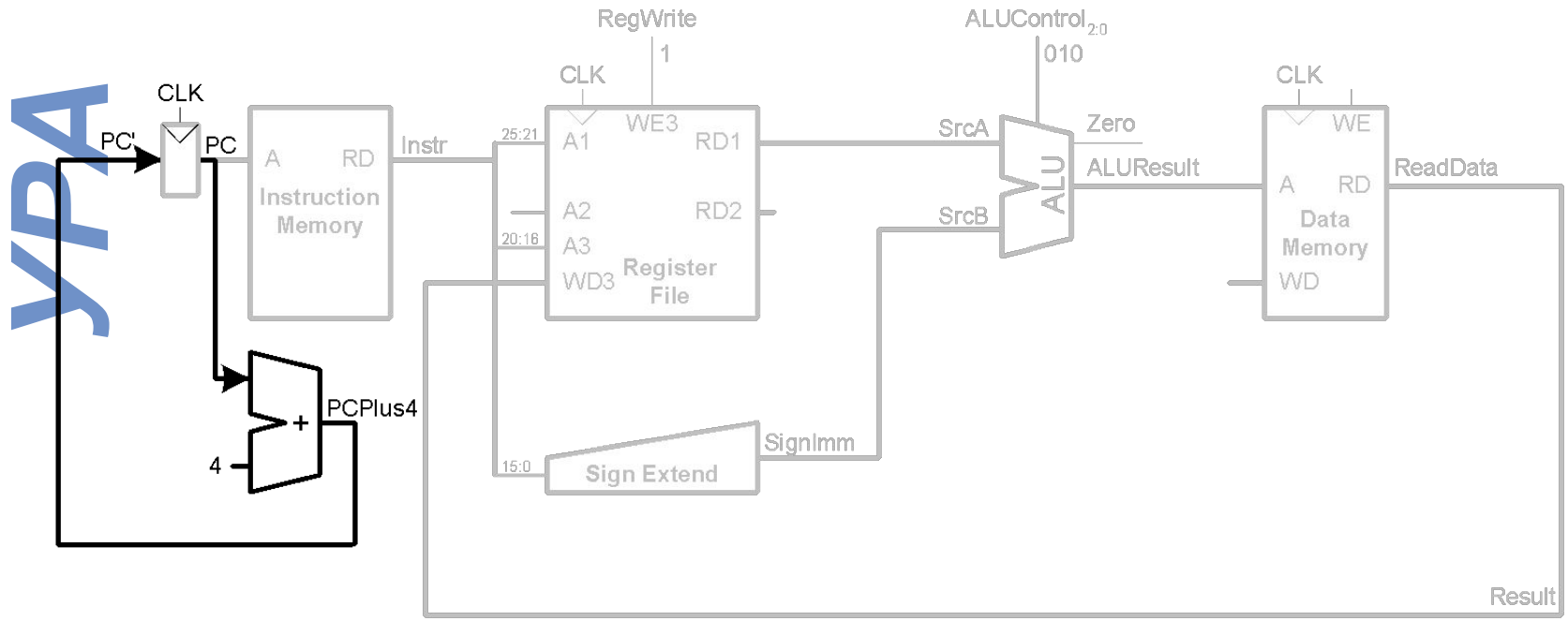

```
lw    rt, imm(rs)
```

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

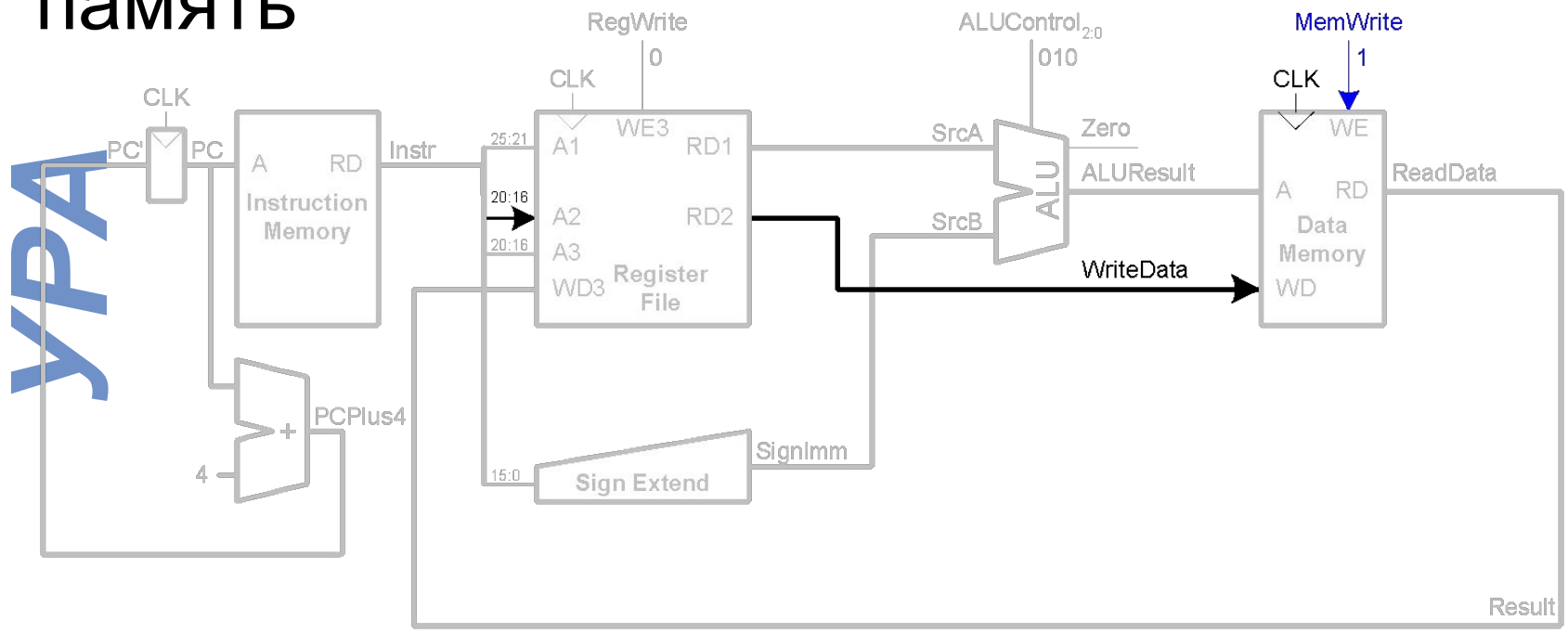
Однотактный тракт данных: увеличение PC

Шаг 6: Вычисляем адрес следующей инструкции



Однотактный тракт данных:

Запись содержимого регистра `rt` в память



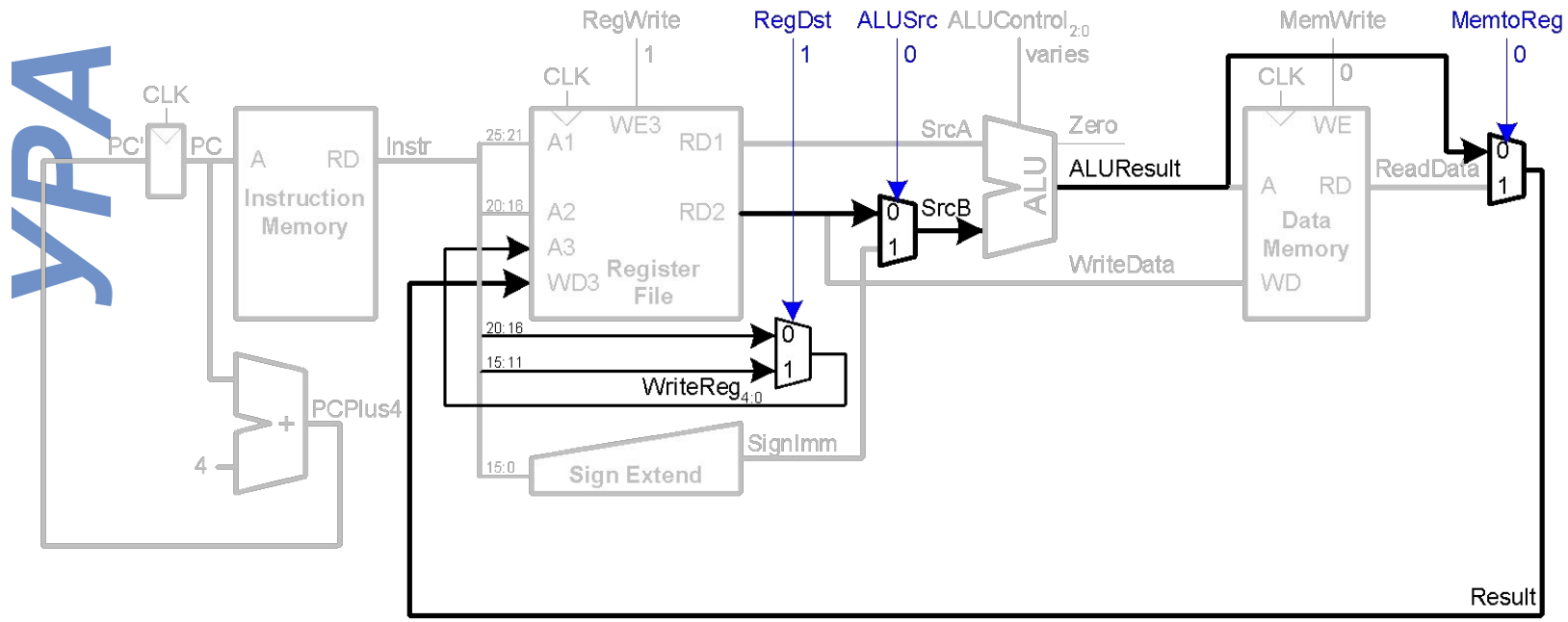

```
sw   rt, imm(rs)
```

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

Однотактный тракт данных: R-

- Считываем операнды из регистров `rs` и `rt`
- Записываем *ALUResult* в регистр с номером из поля `rd` инструкции (для инструкций I-типа результат записывается в регистр с номером `rt`)



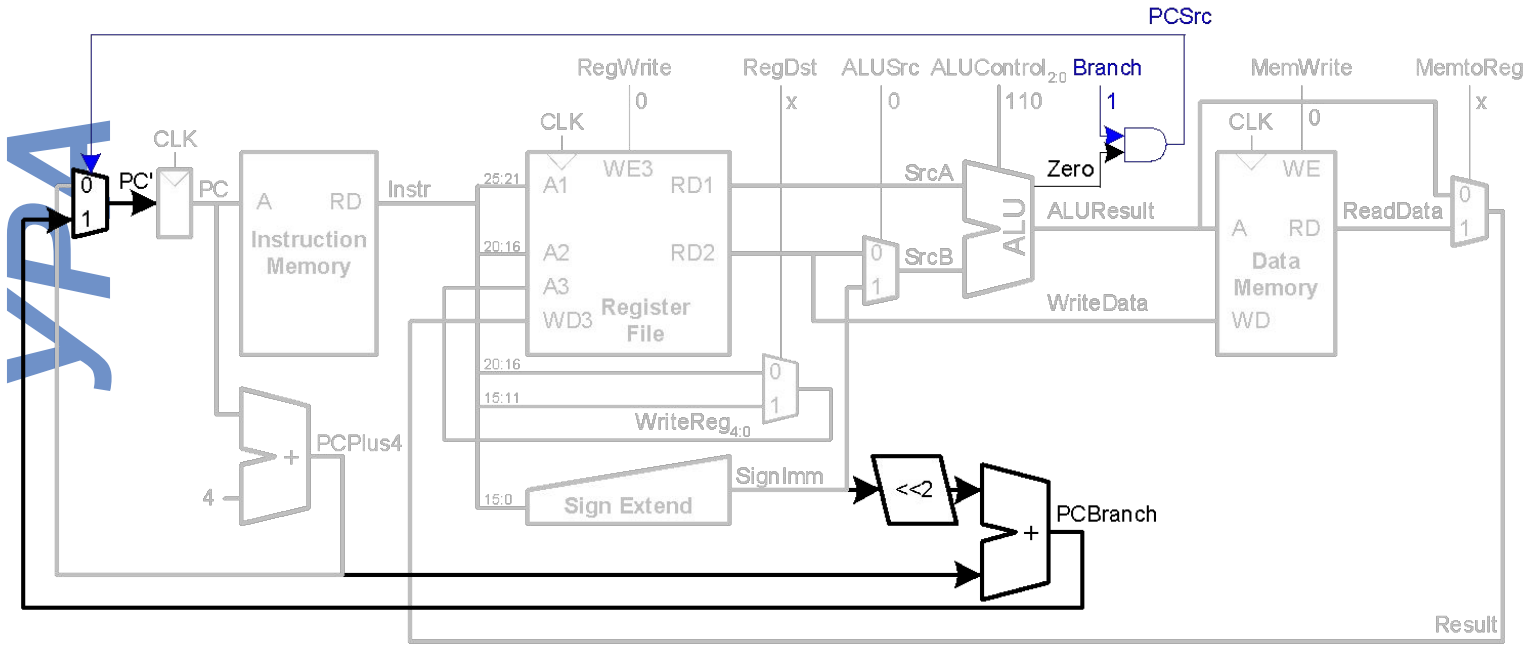
R-Type

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Однотактный тракт данных:

- Проверяем на равенство регистры `rs` и `rt`
- Рассчитываем адрес для условного перехода:

  BTA = (sign-extended immediate << 2) + (PC+4)



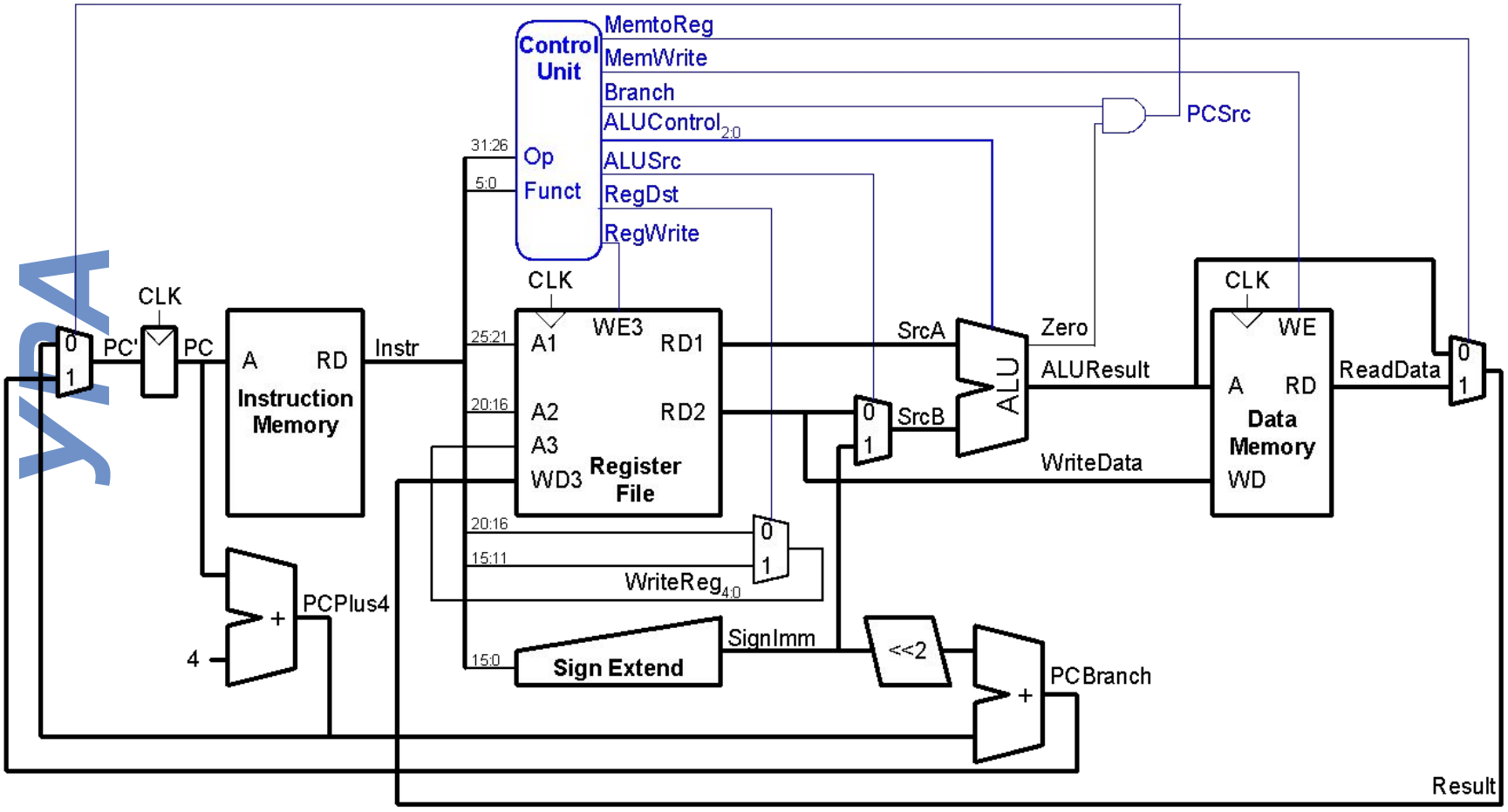
Assembly Code

```
beq $t0, $0, else
(beq $t0, $0, 3)
```

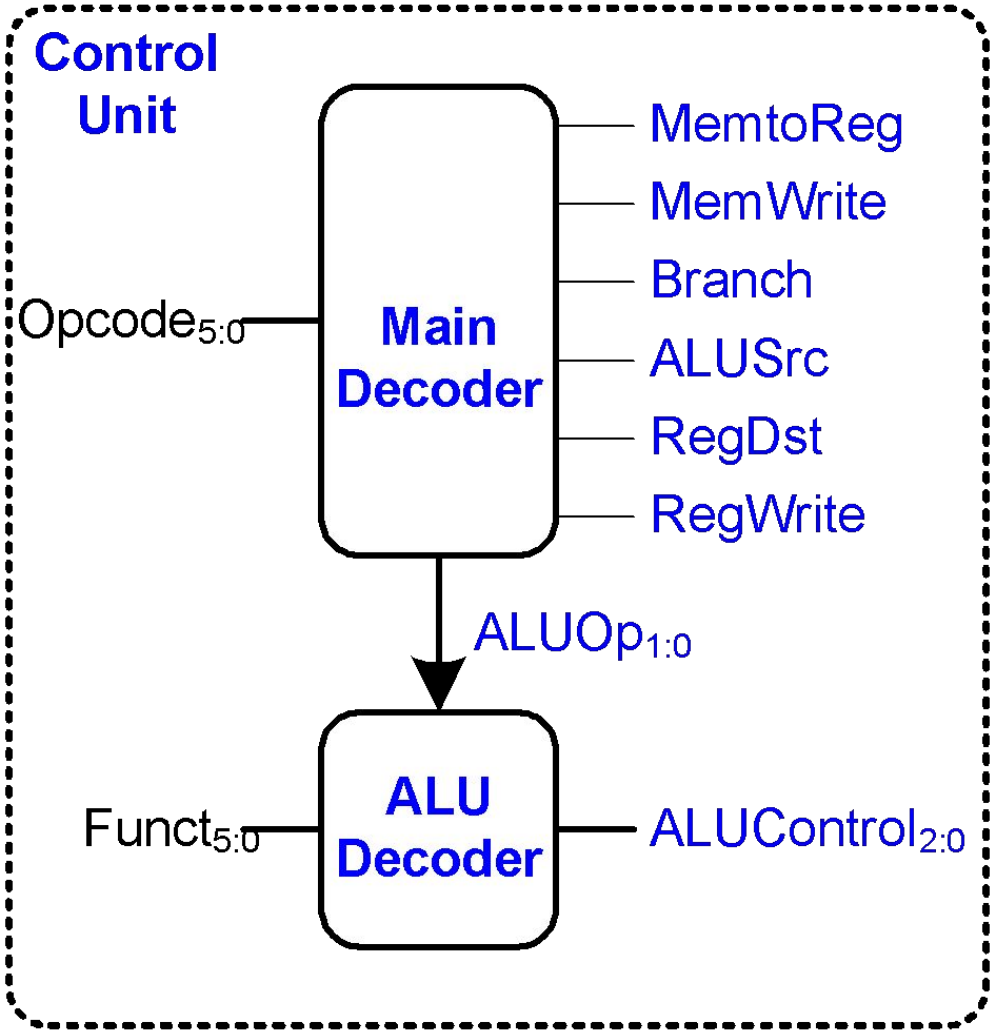
Field Values

op	rs	rt	imm
4	8	0	3
6 bits	5 bits	5 bits	5 bits 5 bits 6 bits

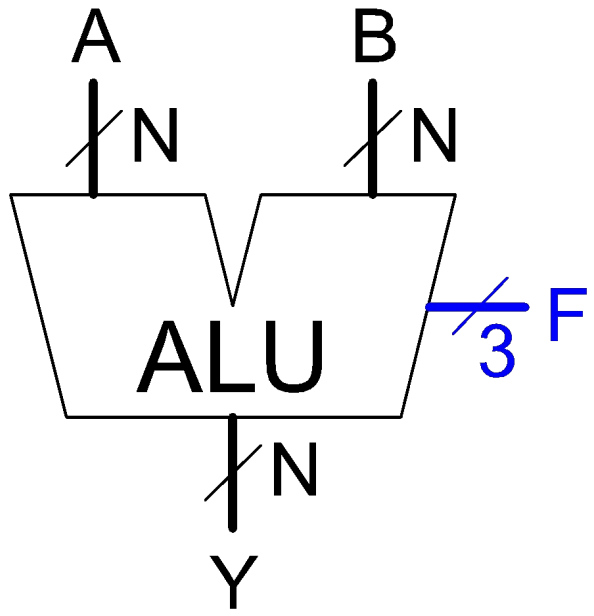
Однотактный процессор



Управление однотактным процессором

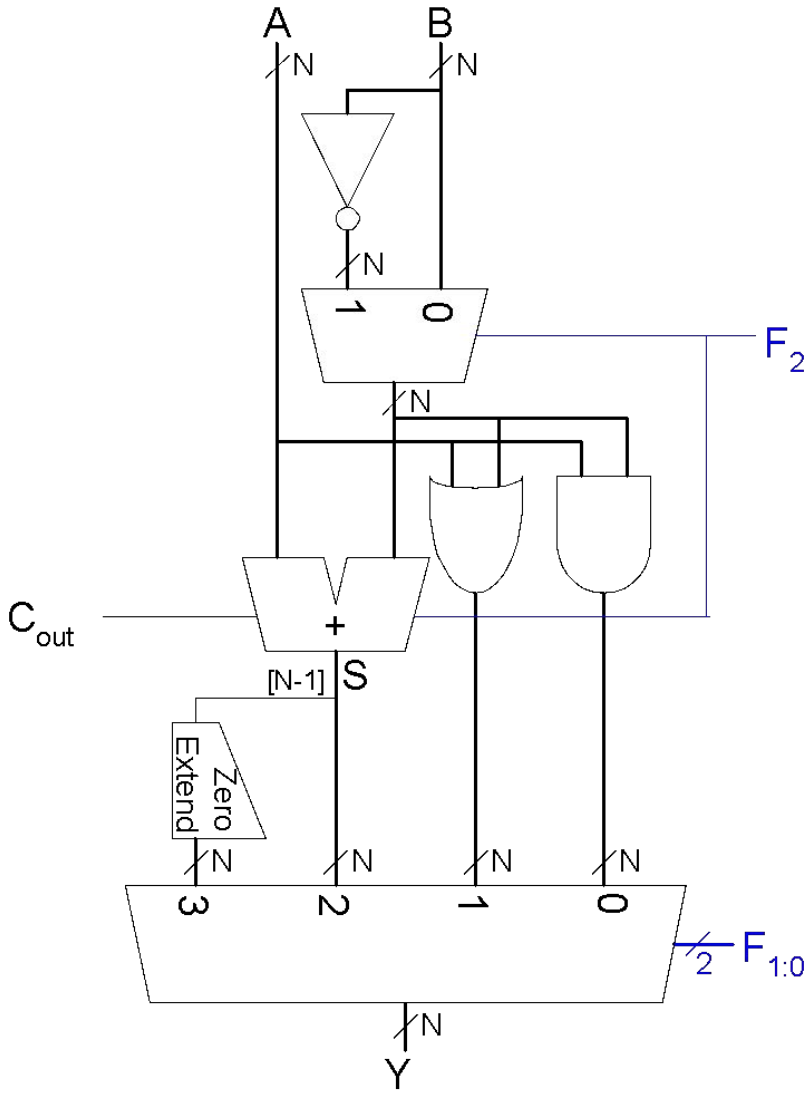


Вспомним принцип работы



$F_{2:0}$	Функция
000	A & B
001	A | B
010	A + B
011	Не исп.
100	A & ~B
101	A | ~B
110	A - B
111	SLT

Вспомним принцип работы



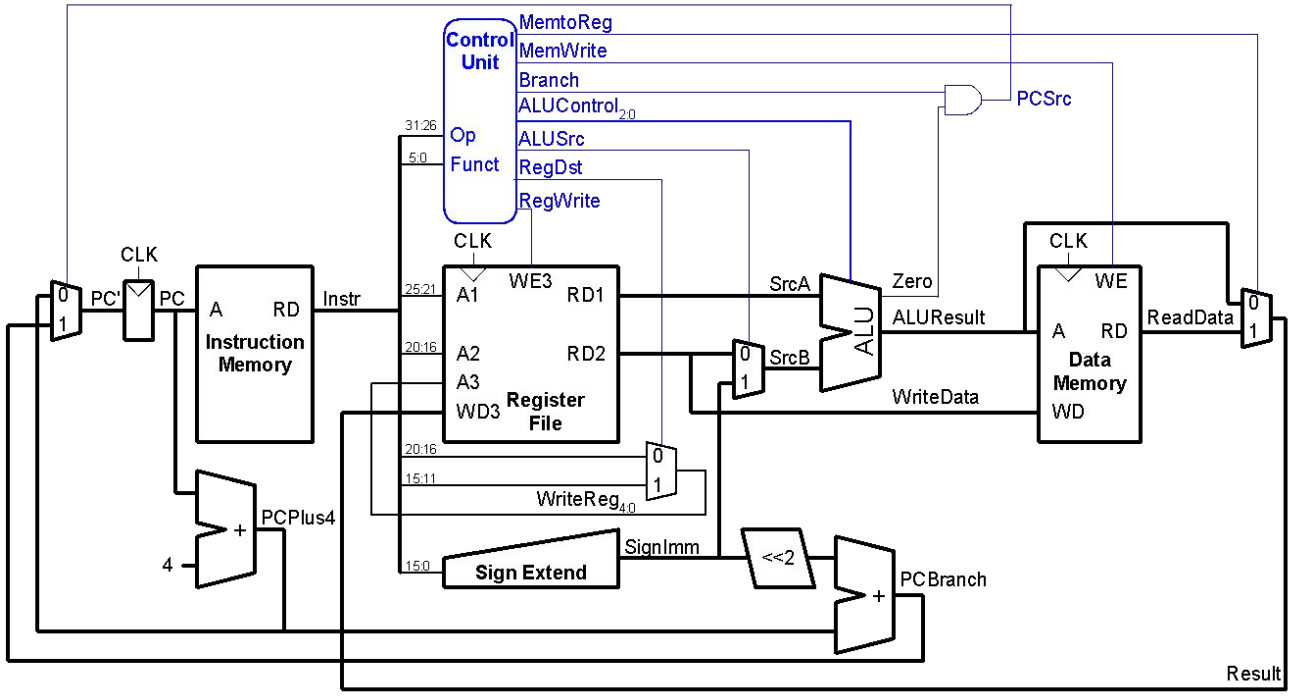
Управляющее устройство: Дешифратор АЛУ

$ALUOp_{1:0}$	Действие
00	Сложение
01	Вычитание
10	Определяется полем Funct
11	Не используется

$ALUOp_{1:0}$	Funct	$ALUControl_{2:0}$
00	X	010 (Сложение)
X1	X	110 (Вычитание)
1X	100000 (`add`)	010 (Сложение)
1X	100010 (`sub`)	110 (Вычитание)
1X	100100 (`and`)	000 (И)
1X	100101 (`or`)	001 (ИЛИ)
1X	101010 (`slt`)	111 (SLT)

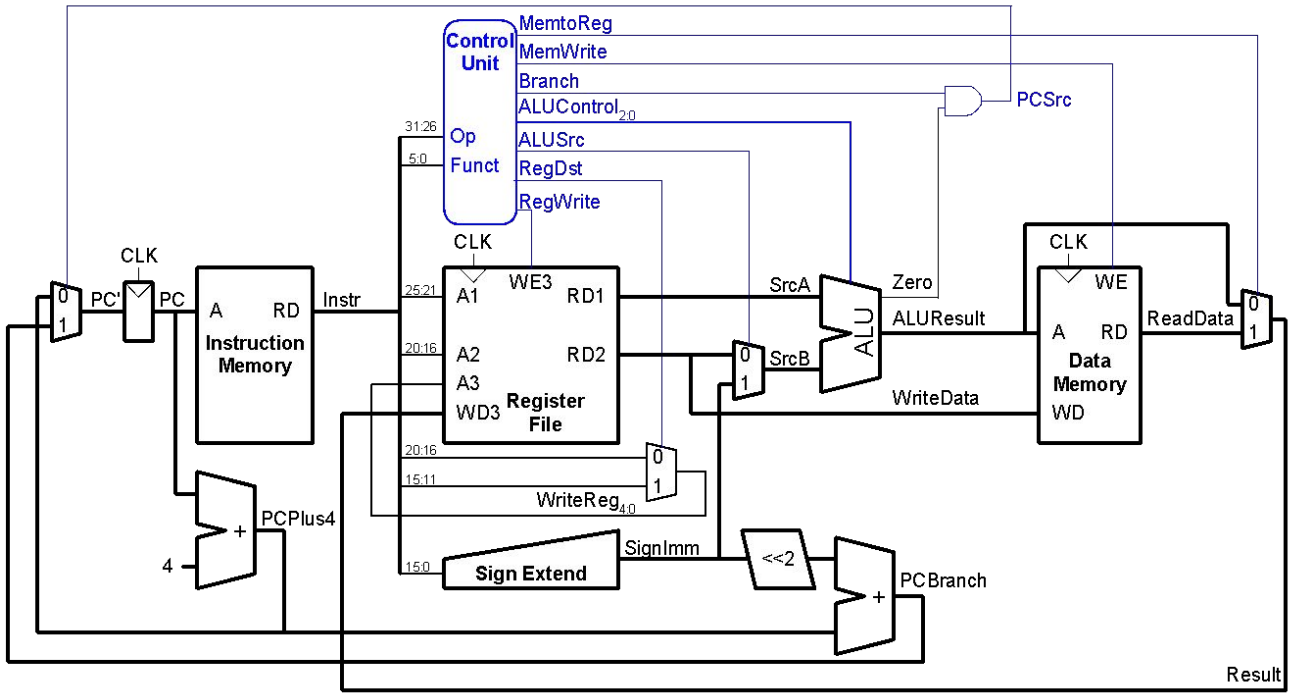
Управляющее устройство: основной дешифратор

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	000000							
`lw`	100011							
`sw`	101011							
`beq`	000100							

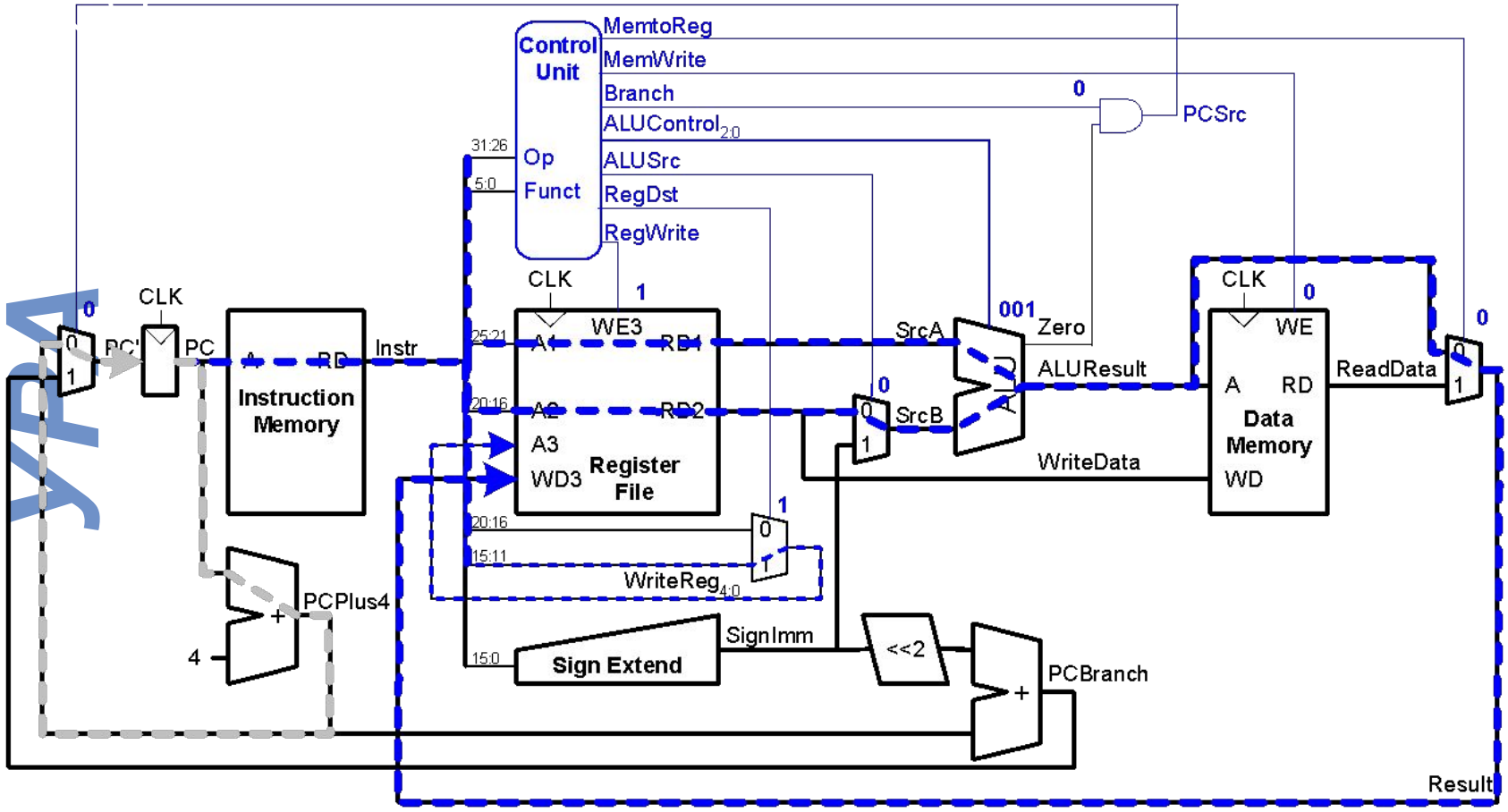


Управляющее устройство: основной дешифратор

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	0	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01



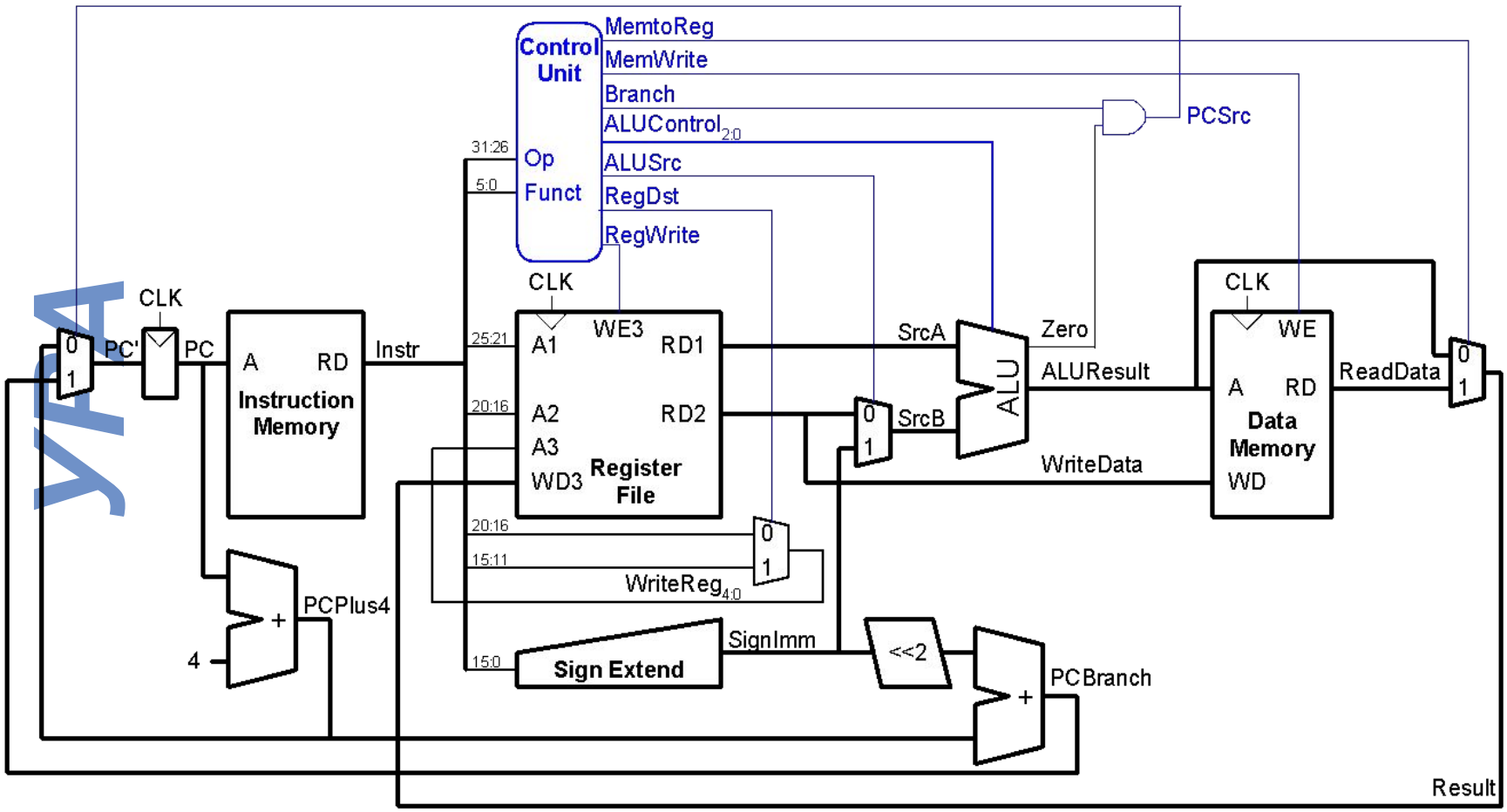
Однотактный тракт данных:



R-Type

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Добавим инструкцию addi



Необходимо сформировать управляющие сигналы, а тракт данных менять не нужно

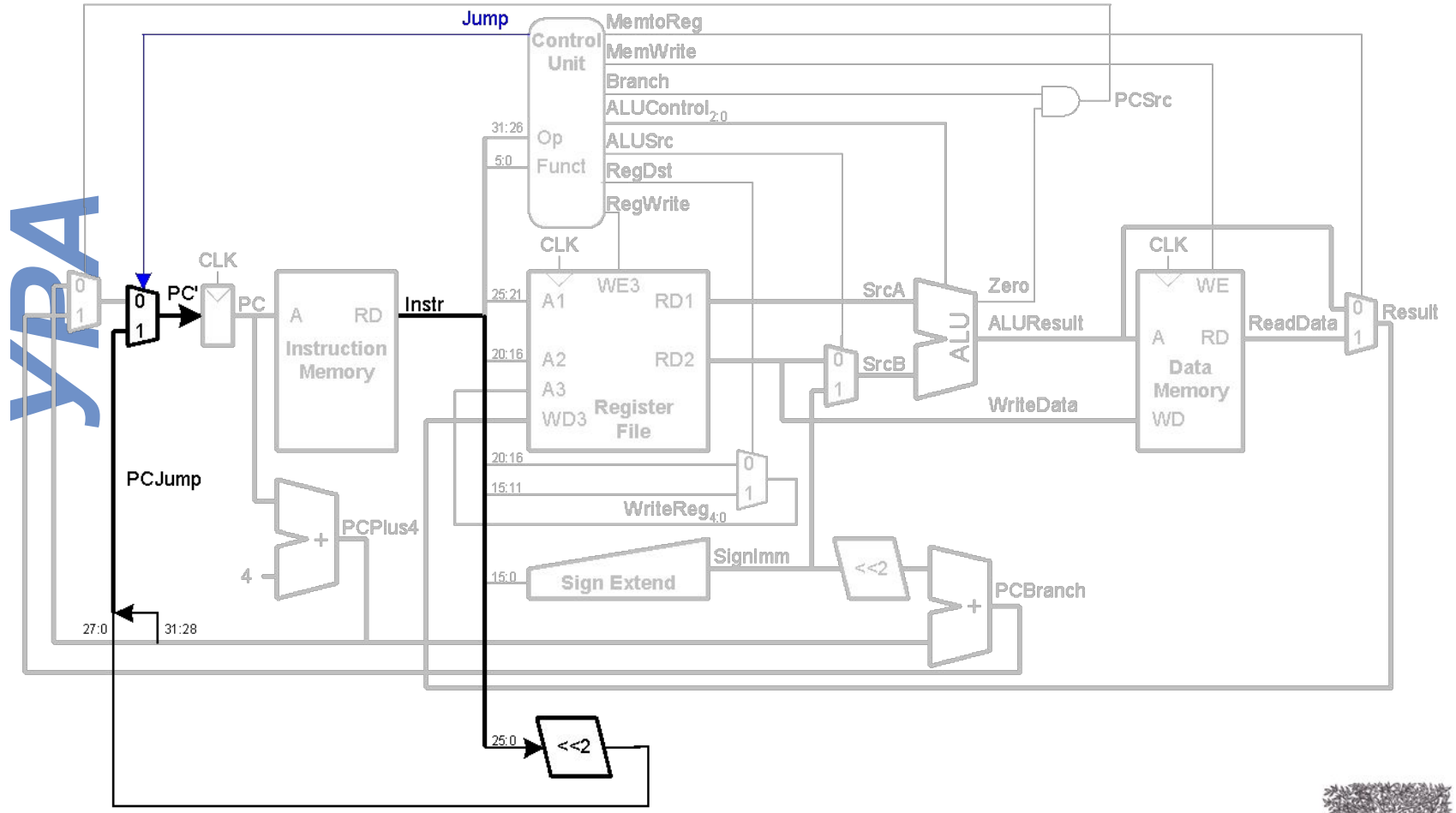
Управляющее устройство:

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	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**							

Управляющее устройство:

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	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**

Добавим функционала: j



Управляющее устройство: j

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump
R-type	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
j	000100								

Управляющее устройство: j

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump
R-type	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**
j	**000100**	**0**	**X**	**X**	**X**	**0**	**X**	**XX**	**1**

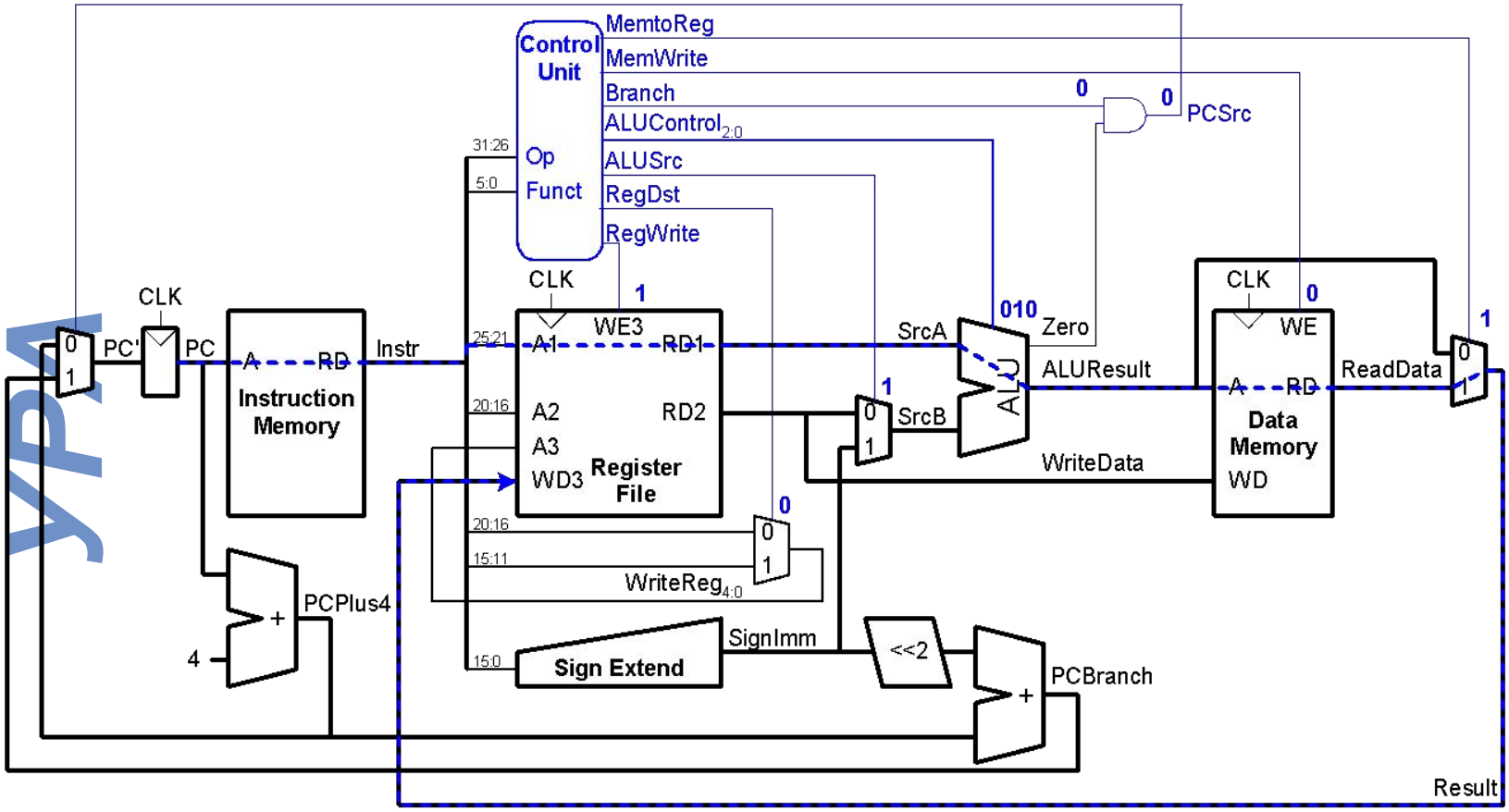
Вернемся к вопросу производительности

Время выполнения программы

= (#инструкции)(такты/инструкция)(секунды/такт)

= # инструкции x CPI x T_c

Производительность однотактного процессора



CPI = 1

T_c определяется цепью с наибольшей задержкой (`lw`)

Производительность однотактного процессора

- Задержка самой длинной цепи комбинационной логики:

$$T_c = t_{pcq\_PC} + t_{\text{mem}} + \max(t_{RF\text{read}}, t_{sext} + t_{\text{mux}}) + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}}$$

- Обычно на длительность периода больше всего влияют:
  - память, АЛУ, регистровый файл
  - $T_c = t_{pcq\_PC} + 2t_{\text{mem}} + t_{RF\text{read}} + t_{\text{mux}} + t_{\text{ALU}} + t_{RF\text{setup}}$

Посчитаем производительность однотактного процессора

Параметр	Обозначение	Задержка (пс)
Время записи в регистр	t_{pcq_PC}	30
Время предустановки регистра	t_{setup}	20
Задержка мультиплексора	t_{mux}	25
Задержка АЛУ	t_{ALU}	200
Задержка считывания из памяти	t_{mem}	250
Задержка считывания из регистрового файла	t_{RFread}	150
Время предустановки регистрового файла	$t_{RFsetup}$	20

$$T_c = ?$$

Посчитаем производительность однотактного процессора

Параметр	Обозначение	Задержка (пс)
Время записи в регистр	t_{pcq_PC}	30
Время предустановки регистра	t_{setup}	20
Задержка мультиплексора	t_{mux}	25
Задержка АЛУ	t_{ALU}	200
Задержка считывания из памяти	t_{mem}	250
Задержка считывания из регистрового файла	t_{RFread}	150
Время предустановки регистрового файла	$t_{RFsetup}$	20

$$T_c = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + t_{mux} + t_{ALU} + t_{RFsetup}$$
$$= [30 + 2(250) + 150 + 25 + 200 + 20] \text{ пс}$$
$$= 925 \text{ пс}$$

Посчитаем производительность однотактного процессора

Предположим, в программе 100 миллиардов инструкций:

Время выполнения = # инструкции x CPI x T_C

$= (100 \times 10^9)(1)(925 \times 10^{-12}\ \text{с})$

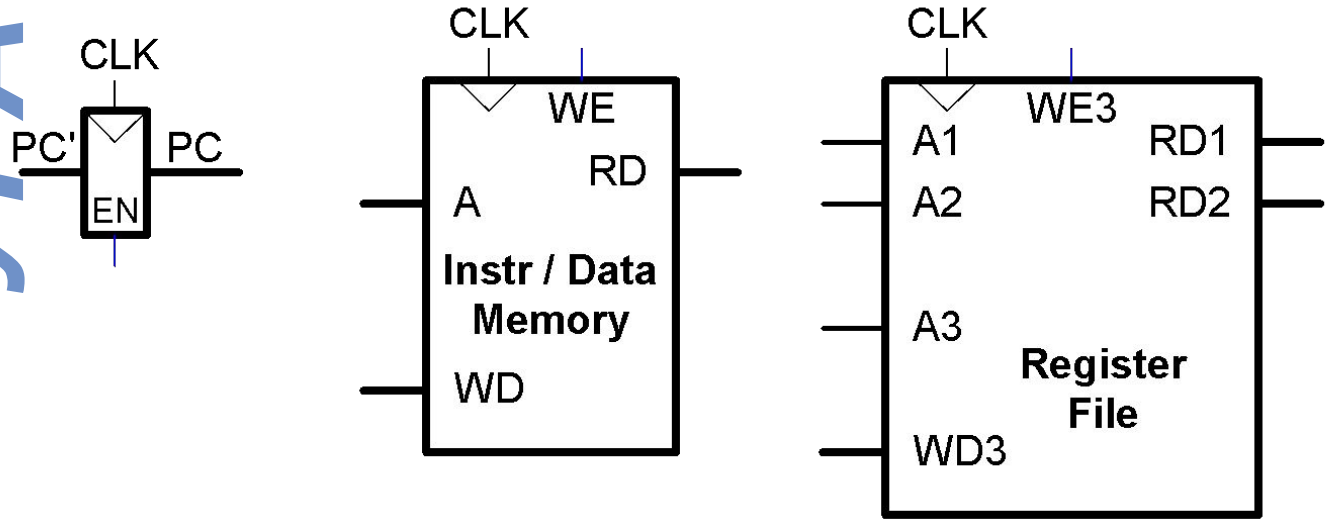
$=$ **92.5 секунд**

Многотактный MIPS

- **Однотактный:**
  - + Простой
  - - Период тактовой частоты ограничен инструкцией с самой длинной цепью комбинационной логики (`lw`)
  - - Несколько сумматоров & 2 отдельных памяти
- **Многотактный:**
  - + Выше тактовая частота
  - + Простые инструкции выполняются быстрее (за меньше тактов)
  - + Повторное использование аппаратурных ресурсов в разных тактах
  - - Значительно усложняется устройство управления
- **Этапы разработки: тракт данных и устройство управления**

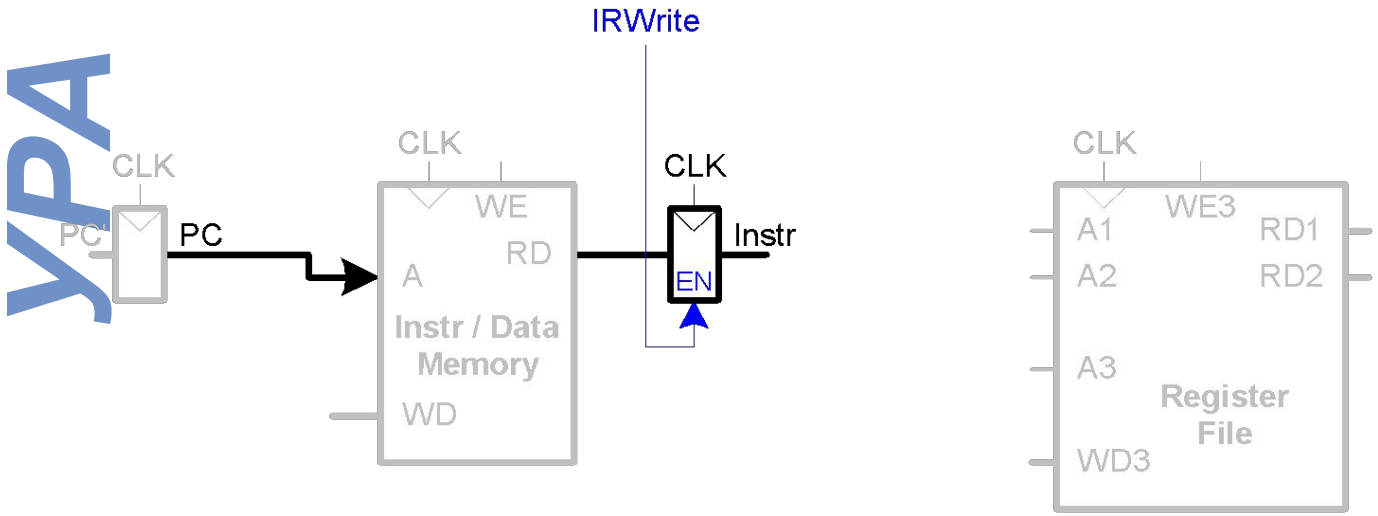
Элементы хранящие состояние многотактного процессора

- Вместо отдельной памяти для инструкций и данных будем использовать одну общую память



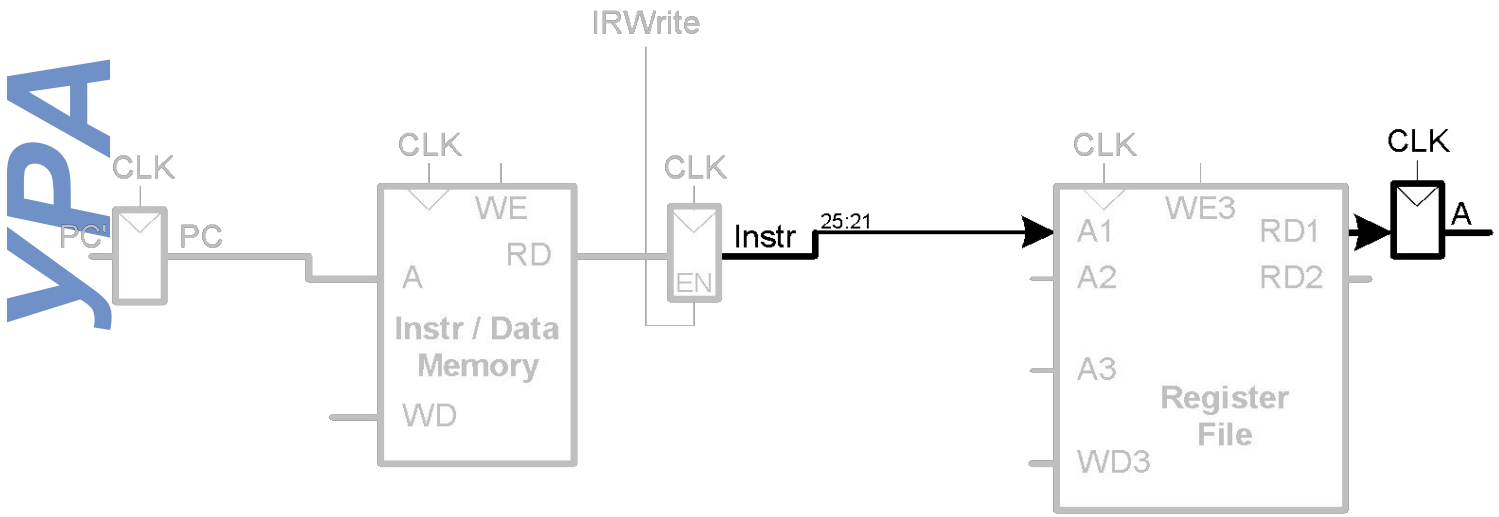
Многотактный тракт данных: Выборка инструкции

Шаг 1: Выборка инструкции



Многотактный тракт данных: чтение регистров

Шаг 2a: считывание операндов-источников из регистрового файла (на примере инструкции `lw`)



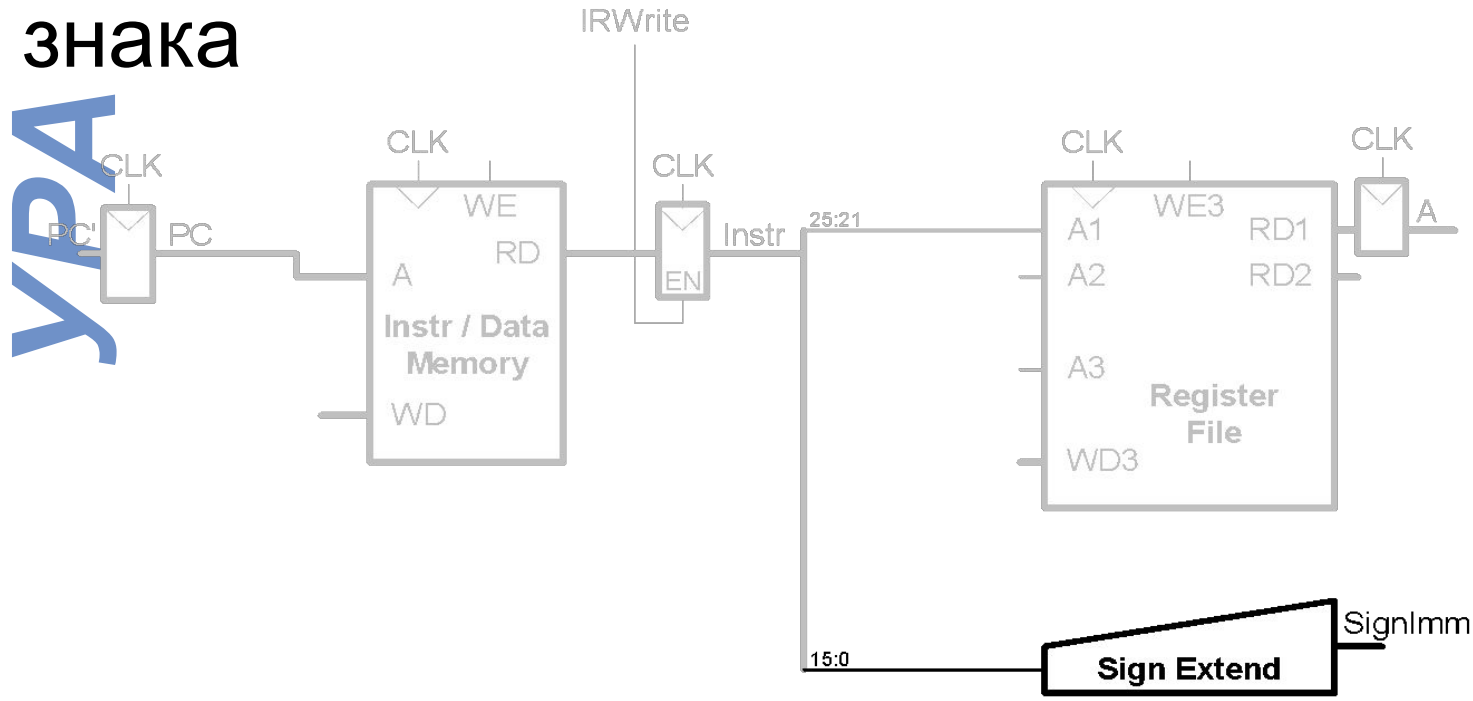

```
lw    rt, imm(rs)
```

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

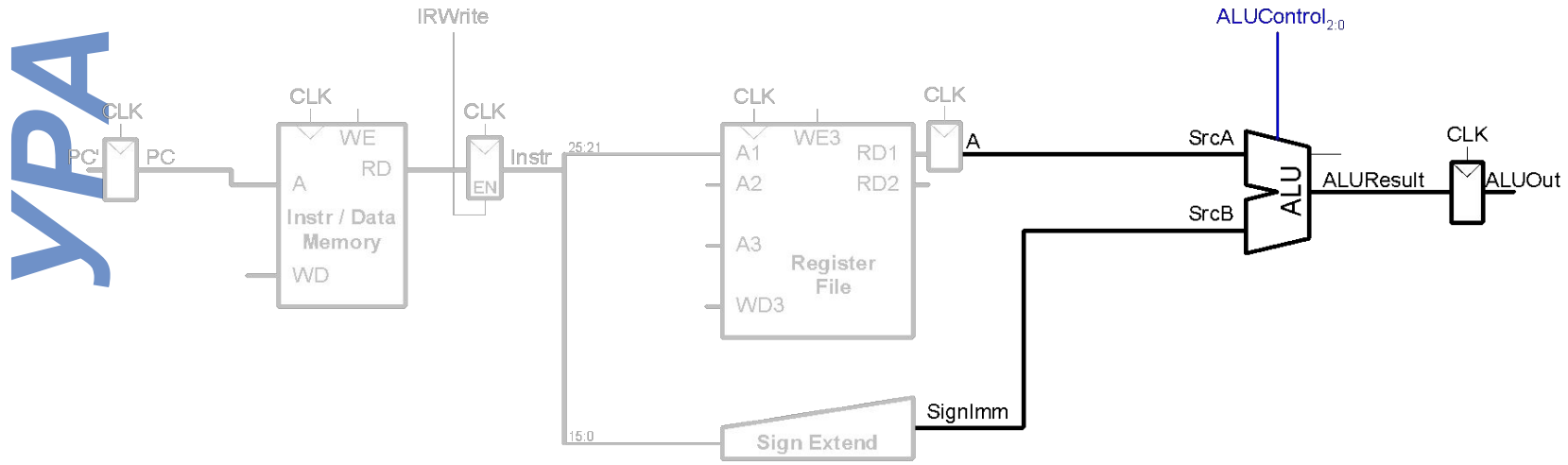
Многотактный тракт данных: расширение константы

Шаг 2b: расширение 16-битной константы до 32-х разрядов битом знака



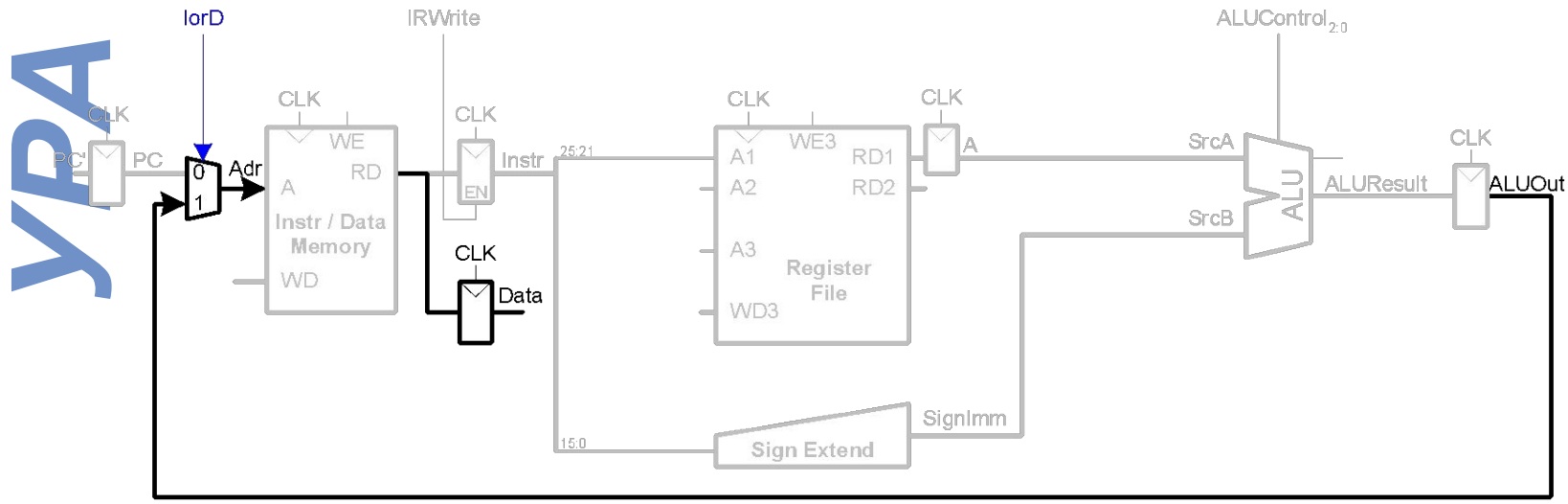
Многотактный тракт данных: вычисление адреса

Шаг 3: Вычисление адреса ячейки в памяти



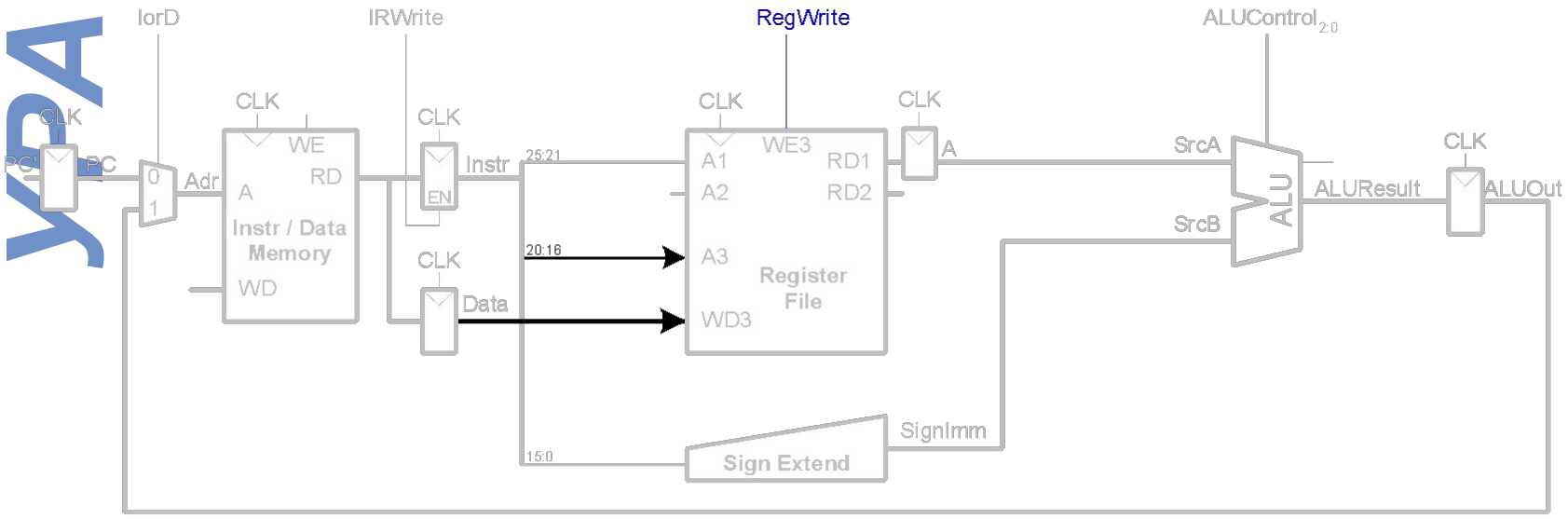
Многотактный тракт данных: считывание из памяти

Шаг 4: считываем данные из памяти



Многотактный тракт данных: запись в регистр

Шаг 5: записываем считанное из памяти 32-битное число в регистр общего назначения, номер которого хранится в поле `rt` инструкции



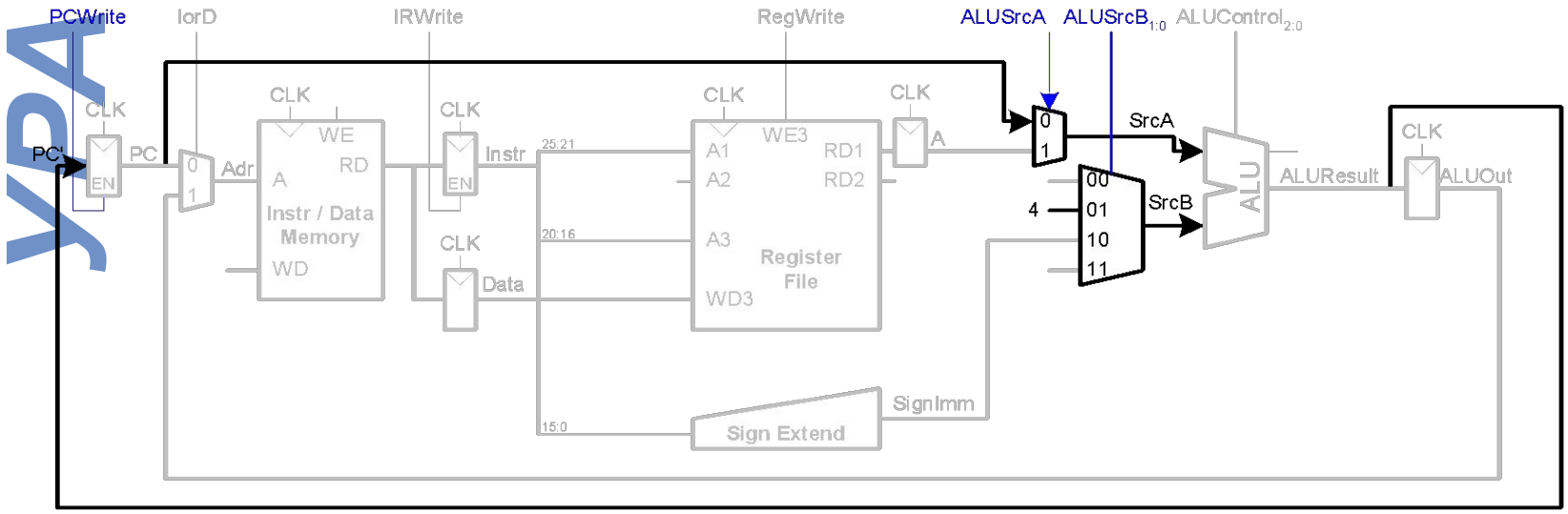

```
lw    rt, imm(rs)
```

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

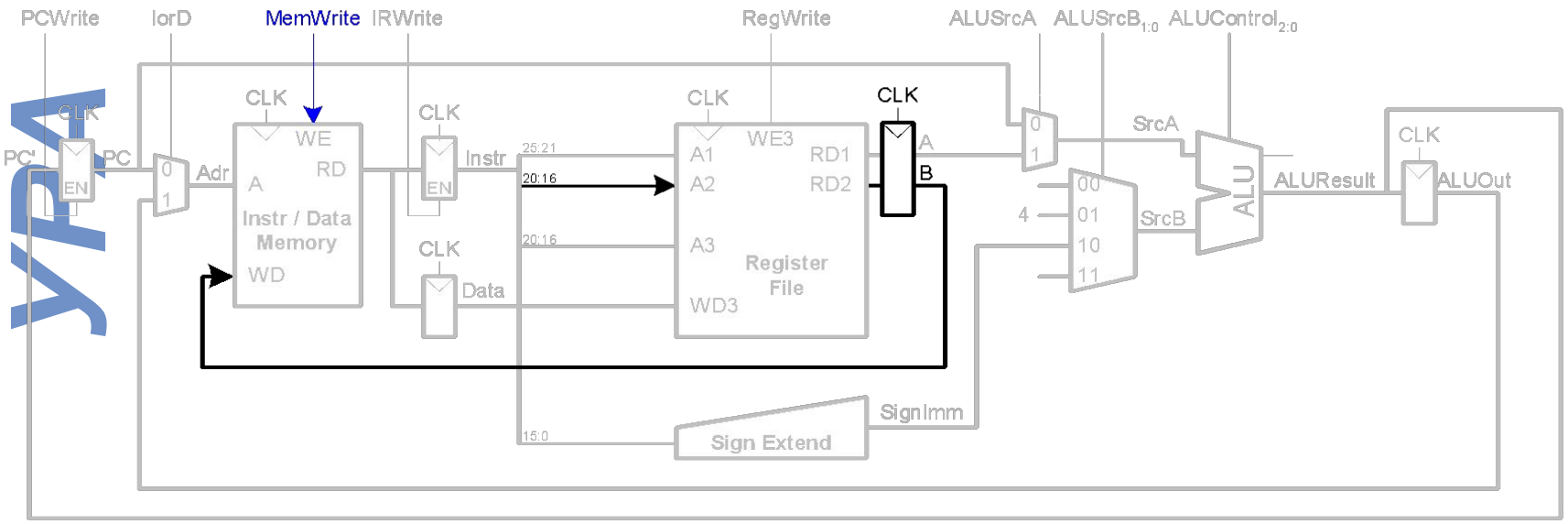
Многотактный тракт данных: увеличиваем PC

Шаг 6: вычисляем адрес следующей инструкции и записываем в PC



Многотактный тракт данных:

Запись содержимого регистра `rt` в память



УРА

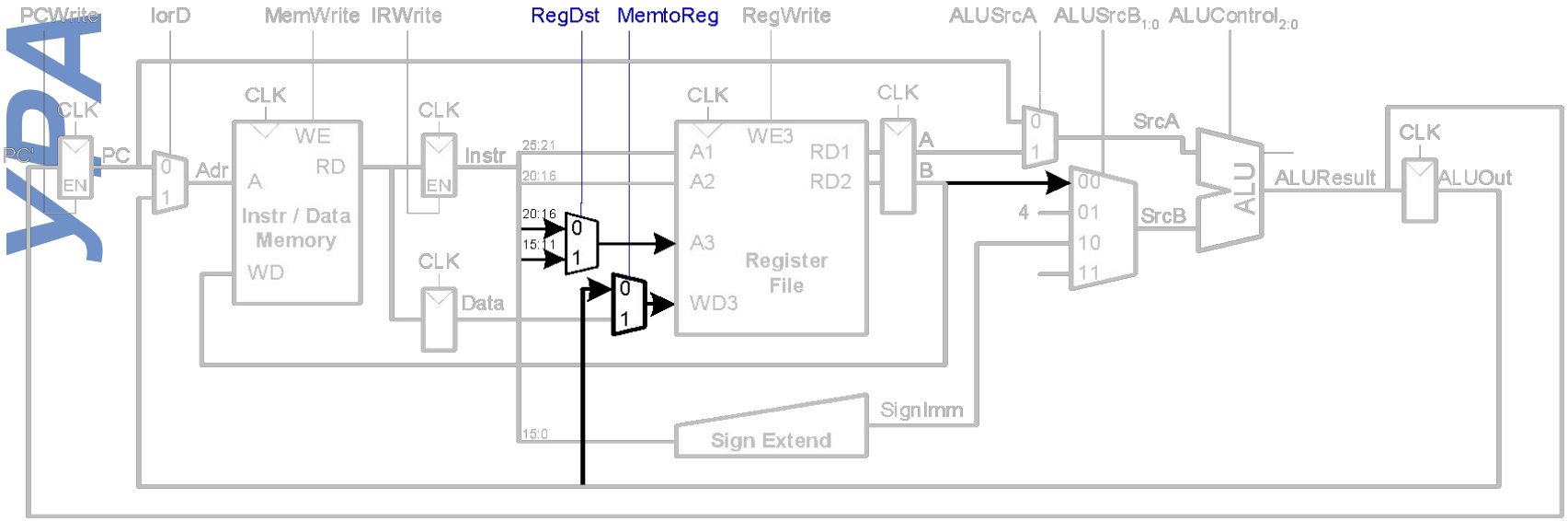
```
sw    rt, imm(rs)
```

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

Многотактный тракт данных: R-

- Считываем операнды из регистров `rs` и `rt`
- Записываем *ALUResult* в регистр с номером из поля `rd` инструкции (для инструкций I-типа результат записывается в регистр с номером `rt`)

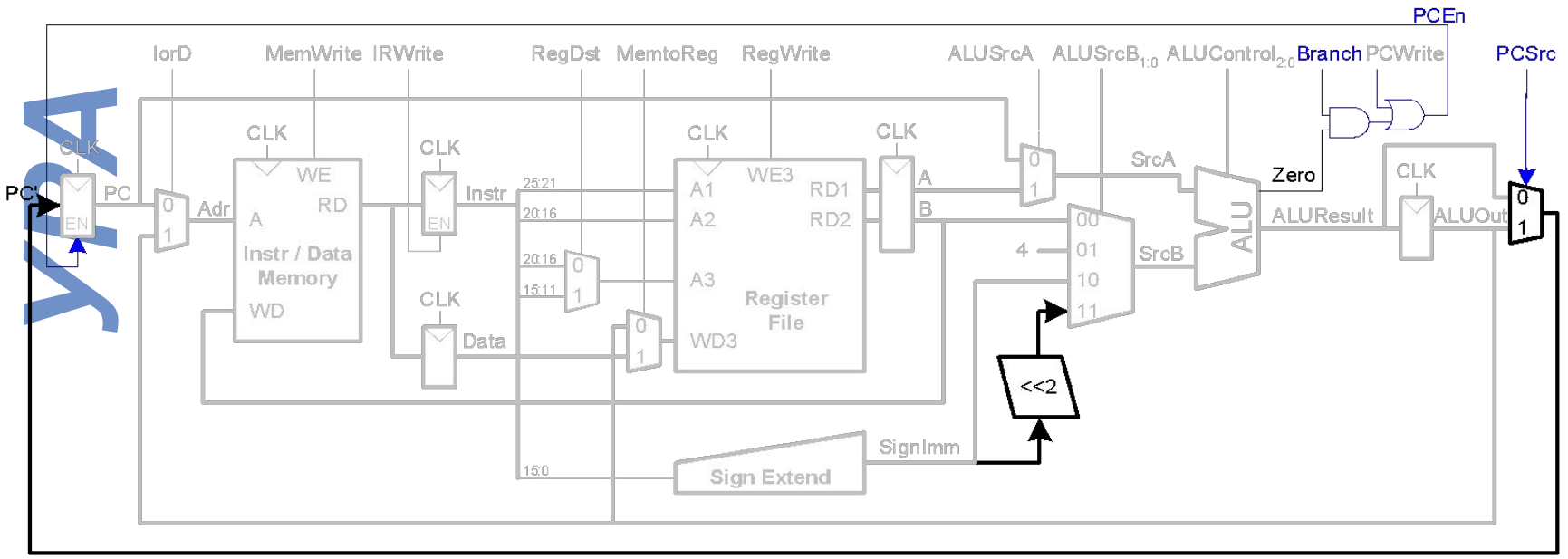


R-Type

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Многотактный тракт данных:

- `rs == rt?`
- BTA = (sign-extended immediate << 2) + (PC+4)



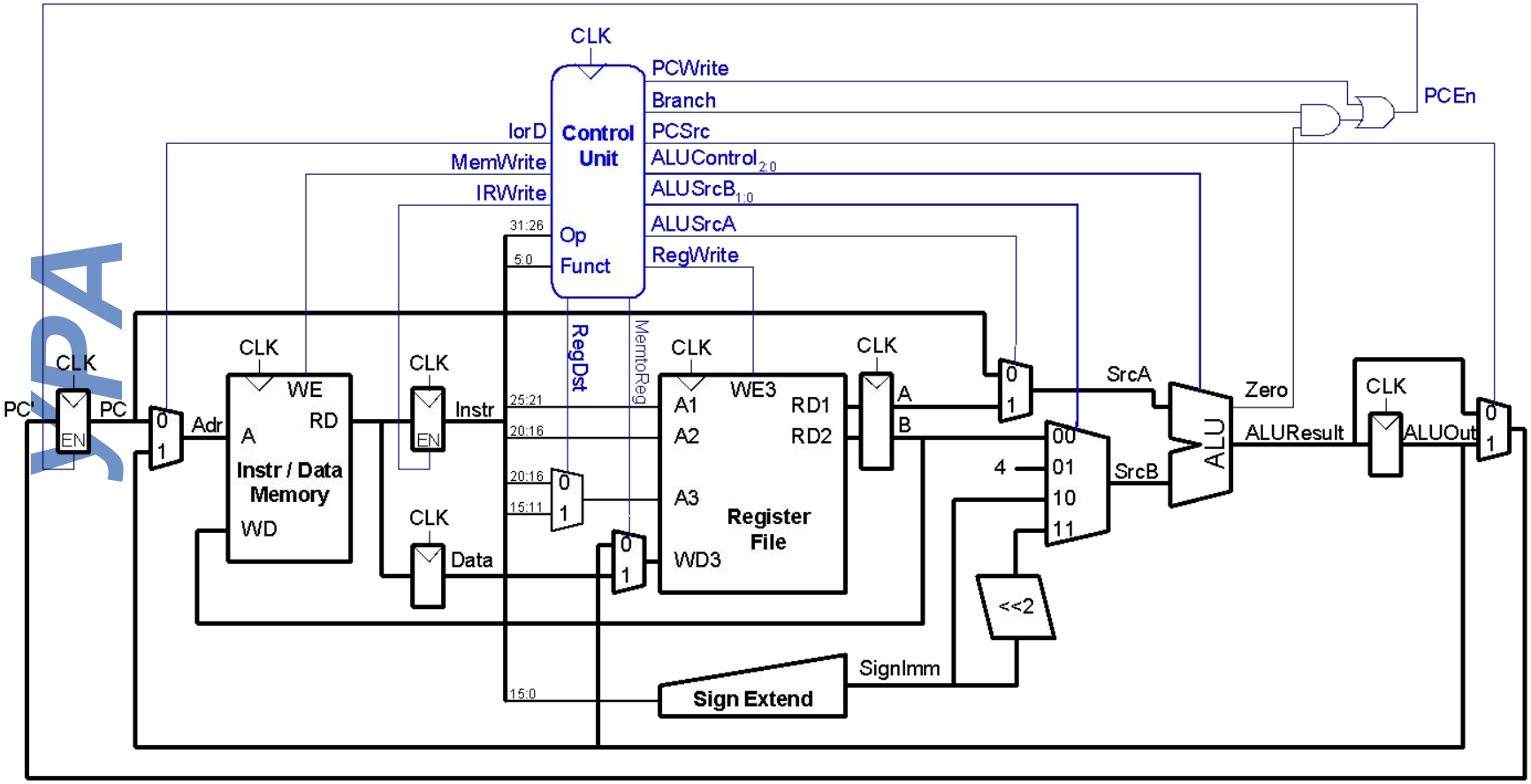
Assembly Code

```
beq $t0, $0, else
(beq $t0, $0, 3)
```

Field Values

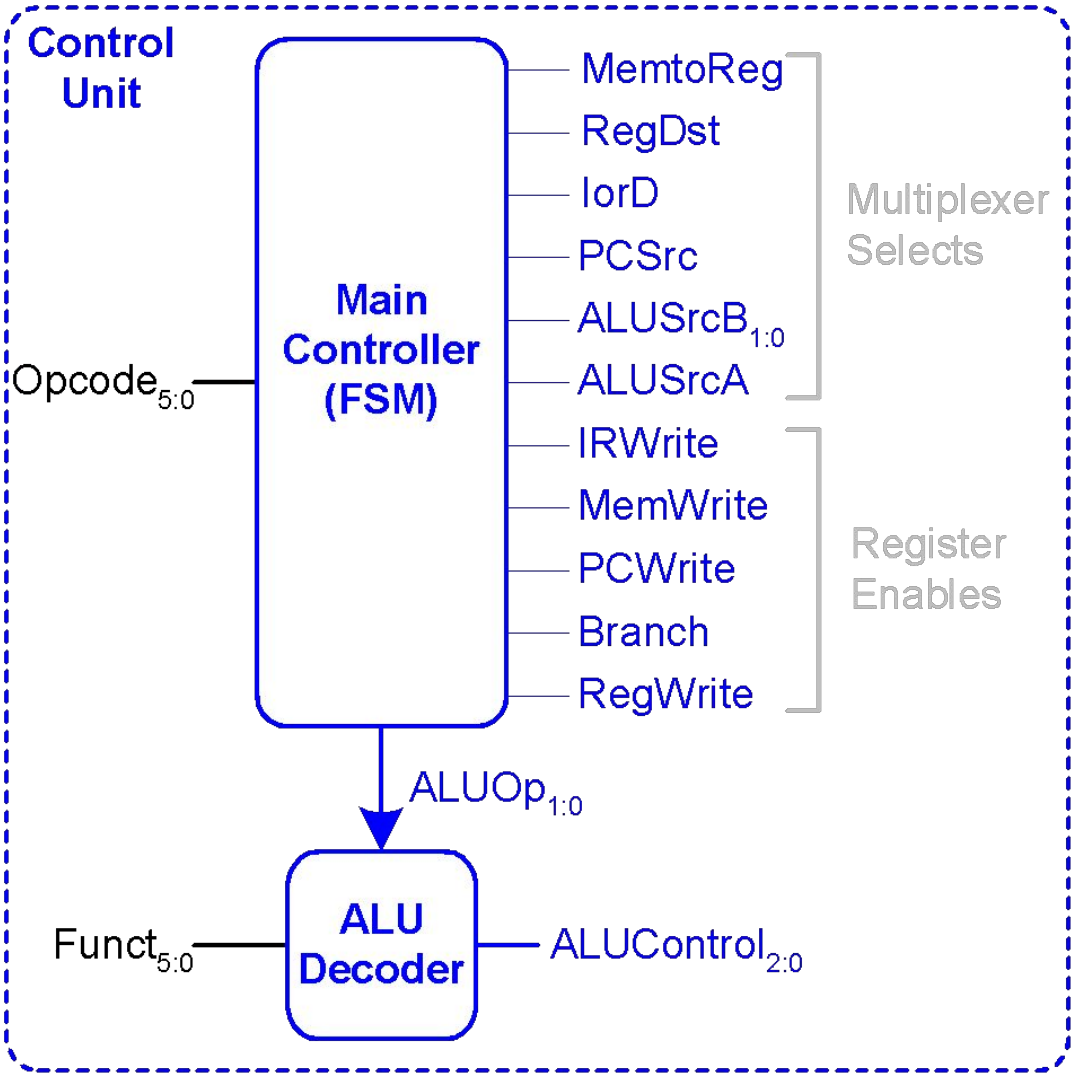
op	rs	rt	imm
4	8	0	3
6 bits	5 bits	5 bits	5 bits 5 bits 6 bits

Многотактный процессор



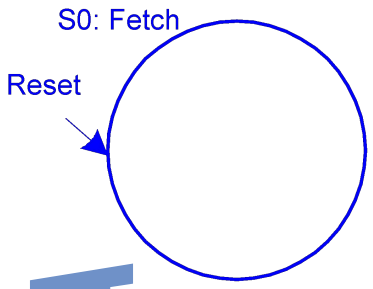
Многотактное устройство управления

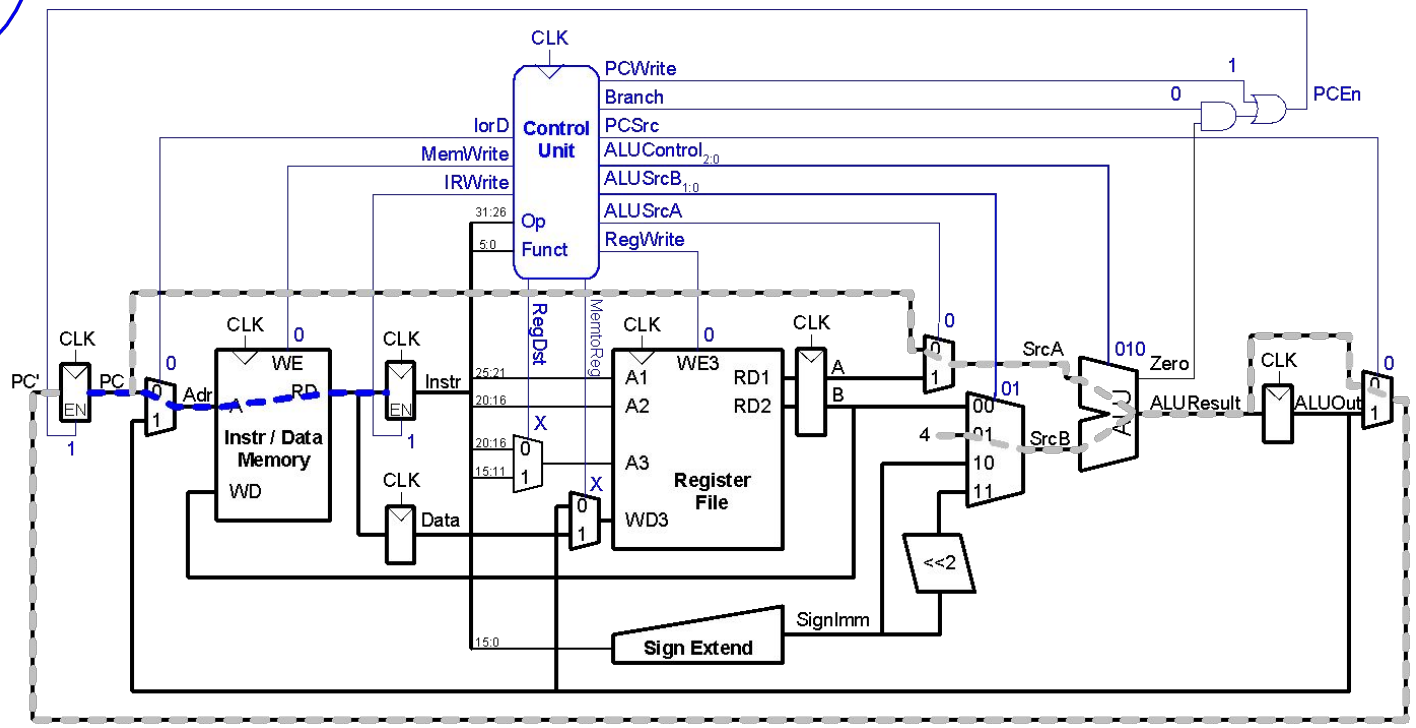
МИКРОАРХИТЕКТУРА



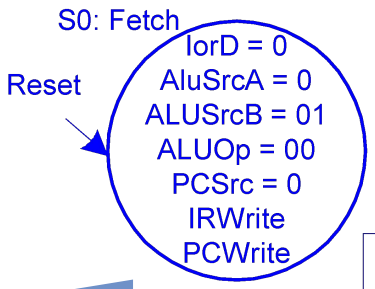
Основной управляющий автомат: Выборка

МИКРОАРХИТЕКТУРА

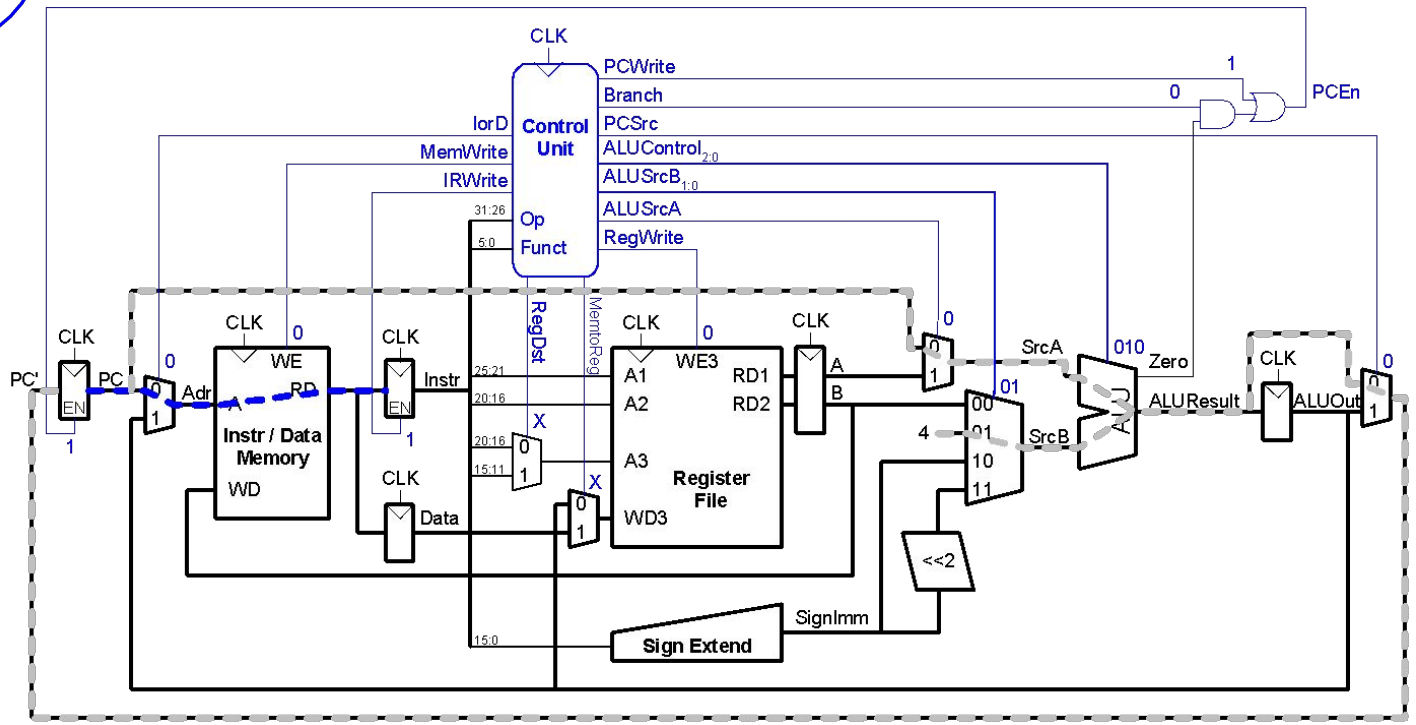




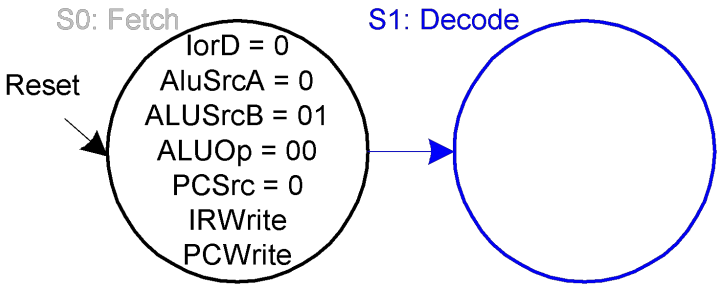
Основной управляющий автомат: Выборка



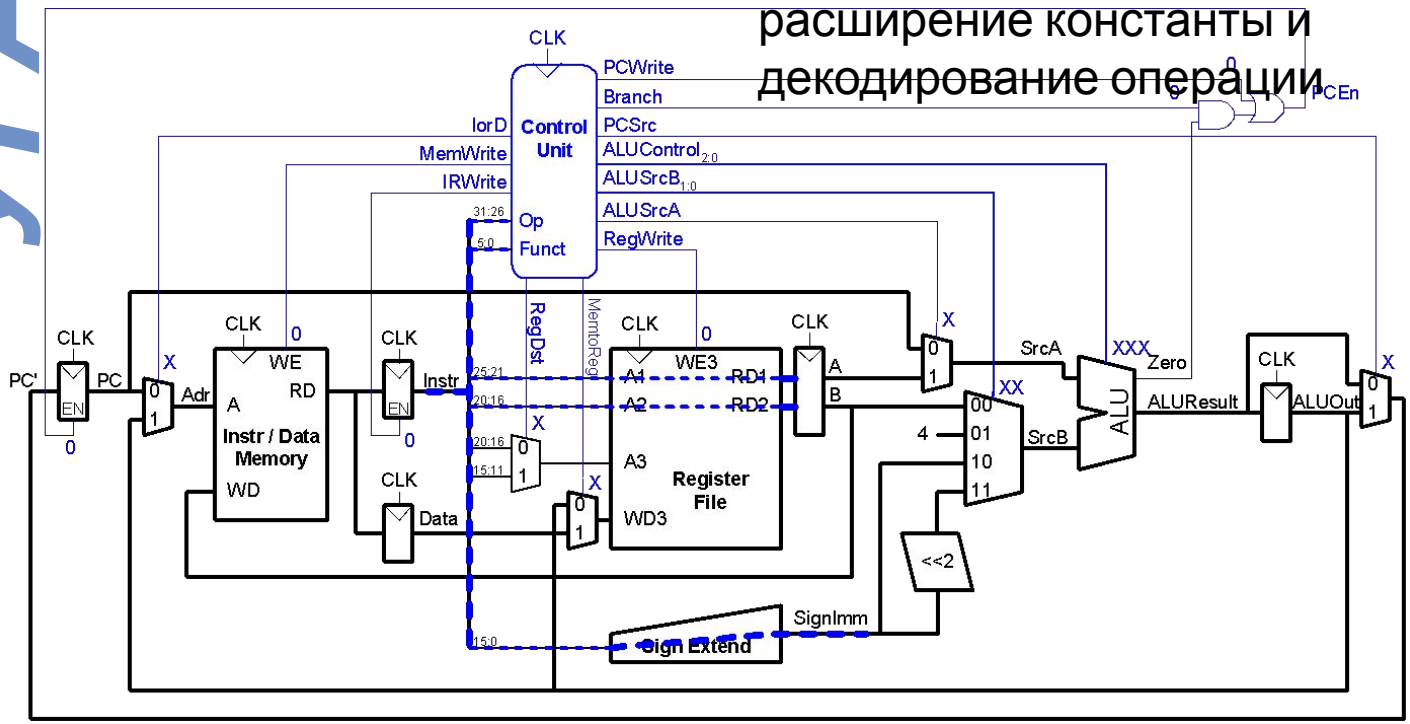
- Сигналы разрешения записи будем показывать только если они не равны нулю
- Одновременно со считыванием инструкции при помощи АЛУ увеличиваем на 4 содержимое PC



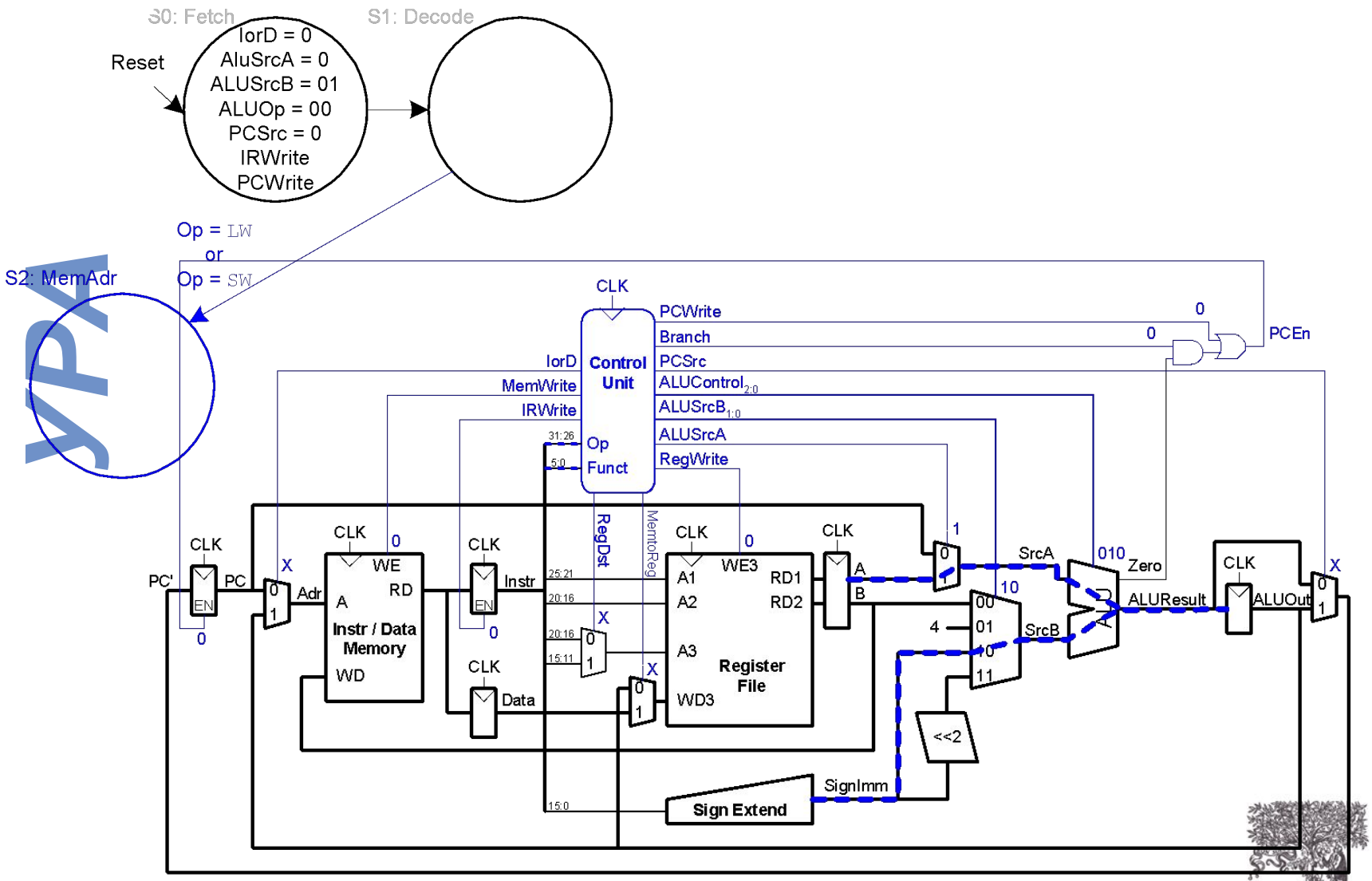
Основной управляющий автомат: Декодирование



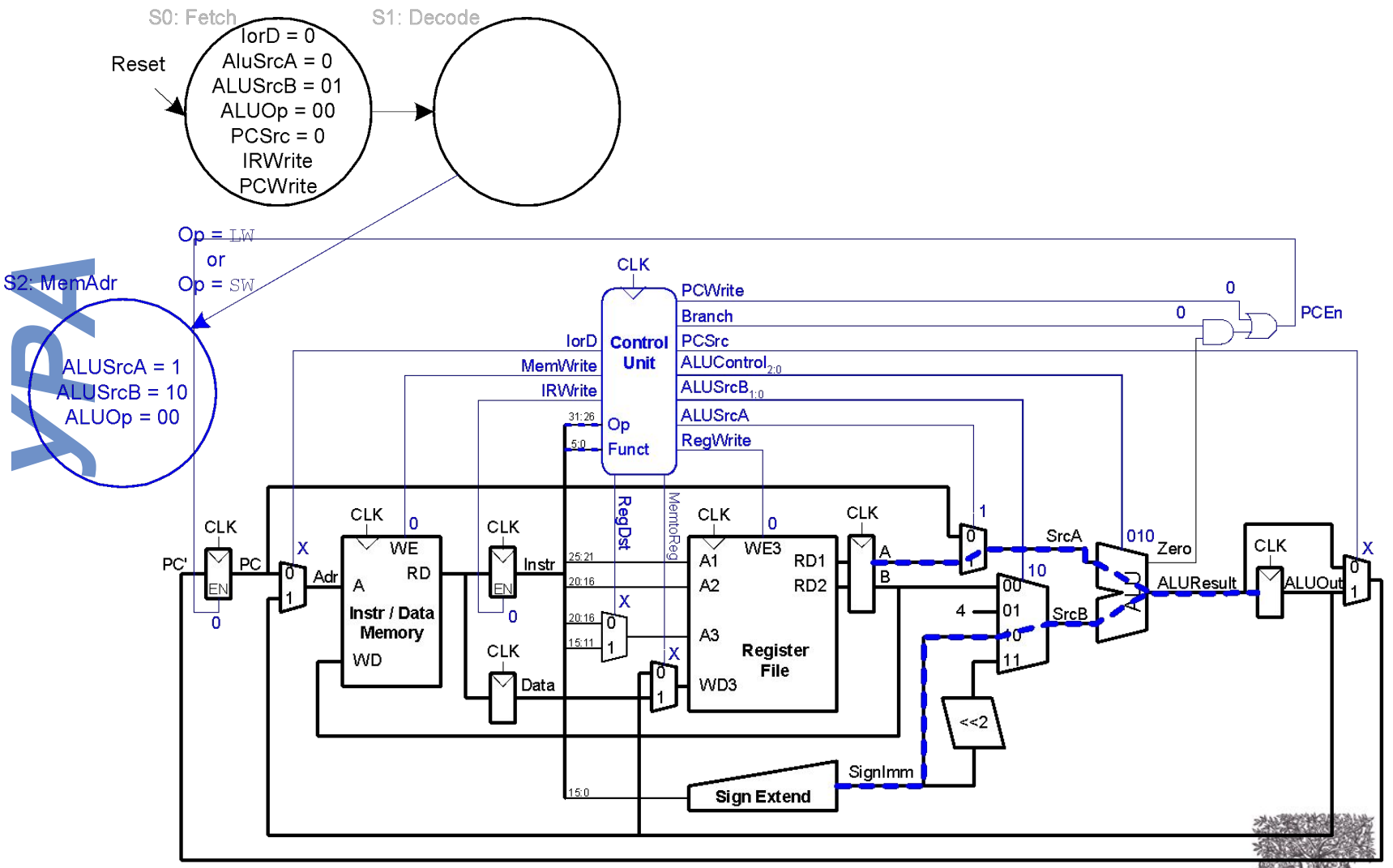
- Будем указывать только те управляющие сигналы, которые имеют смысл на конкретном этапе выполнения команды
- На этом этапе выполняется считывание из регистрового файла, расширение константы и декодирование операции



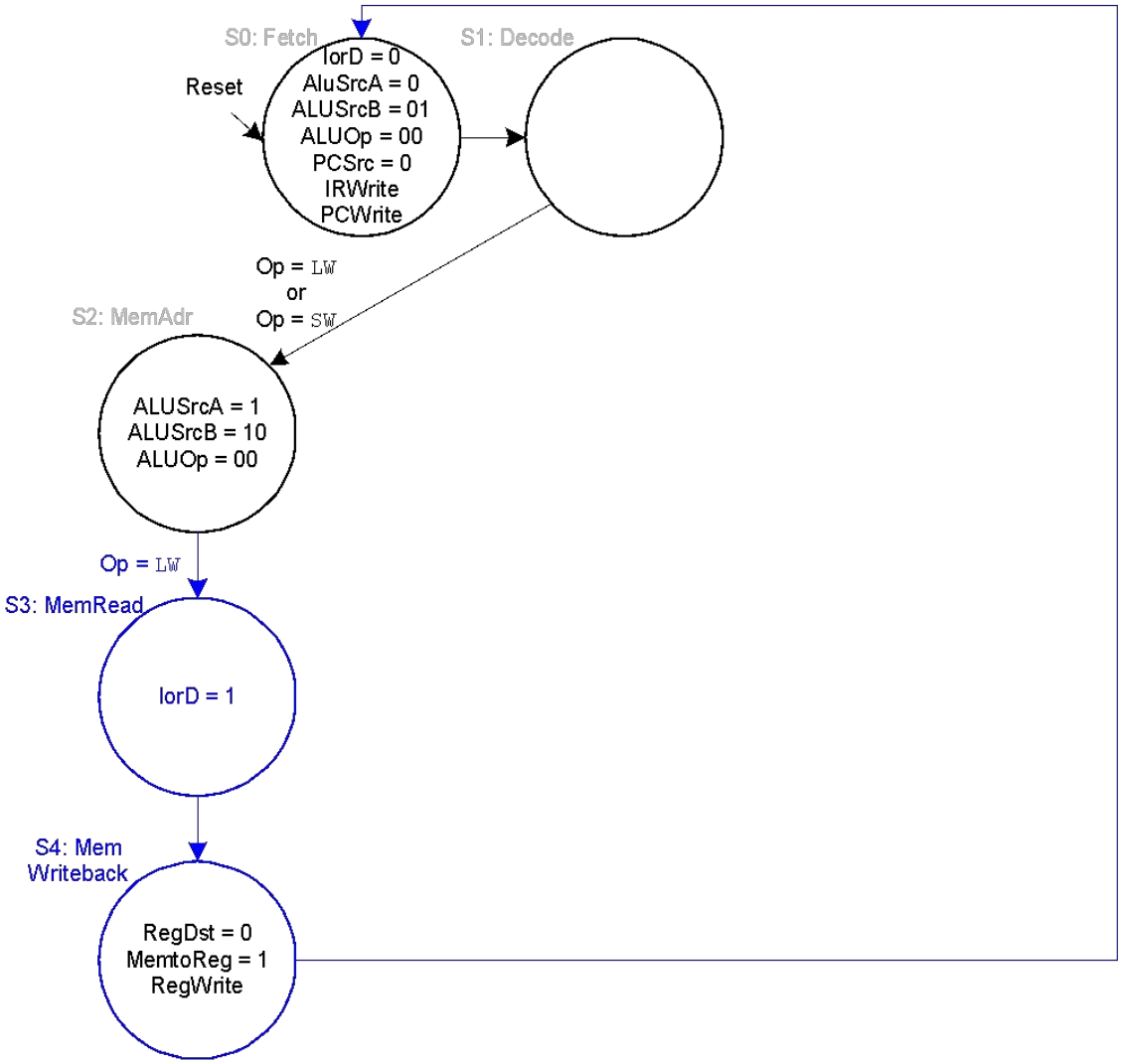
Основной управляющий автомат: Адрес



Основной управляющий автомат: Адрес

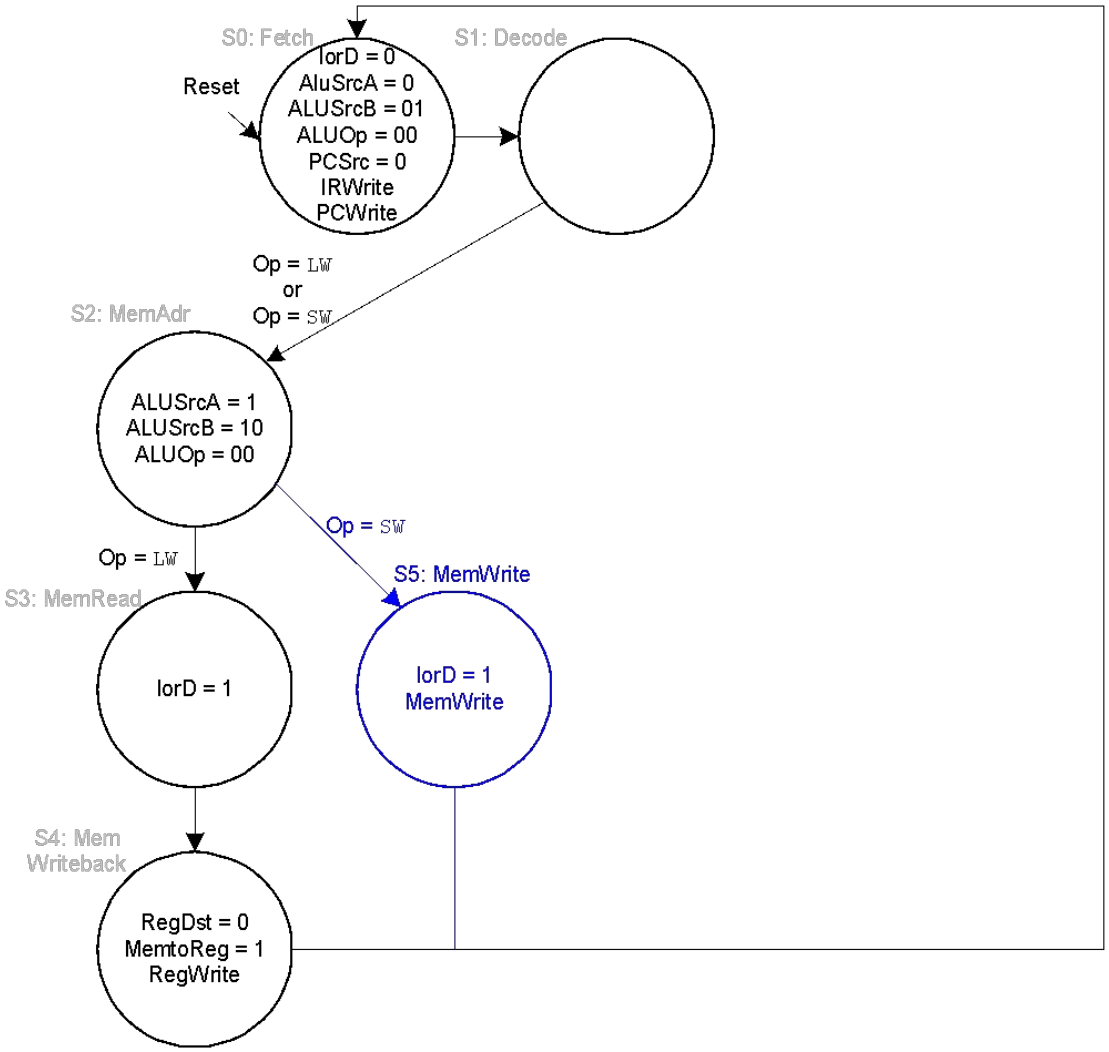


Основной управляющий автомат:

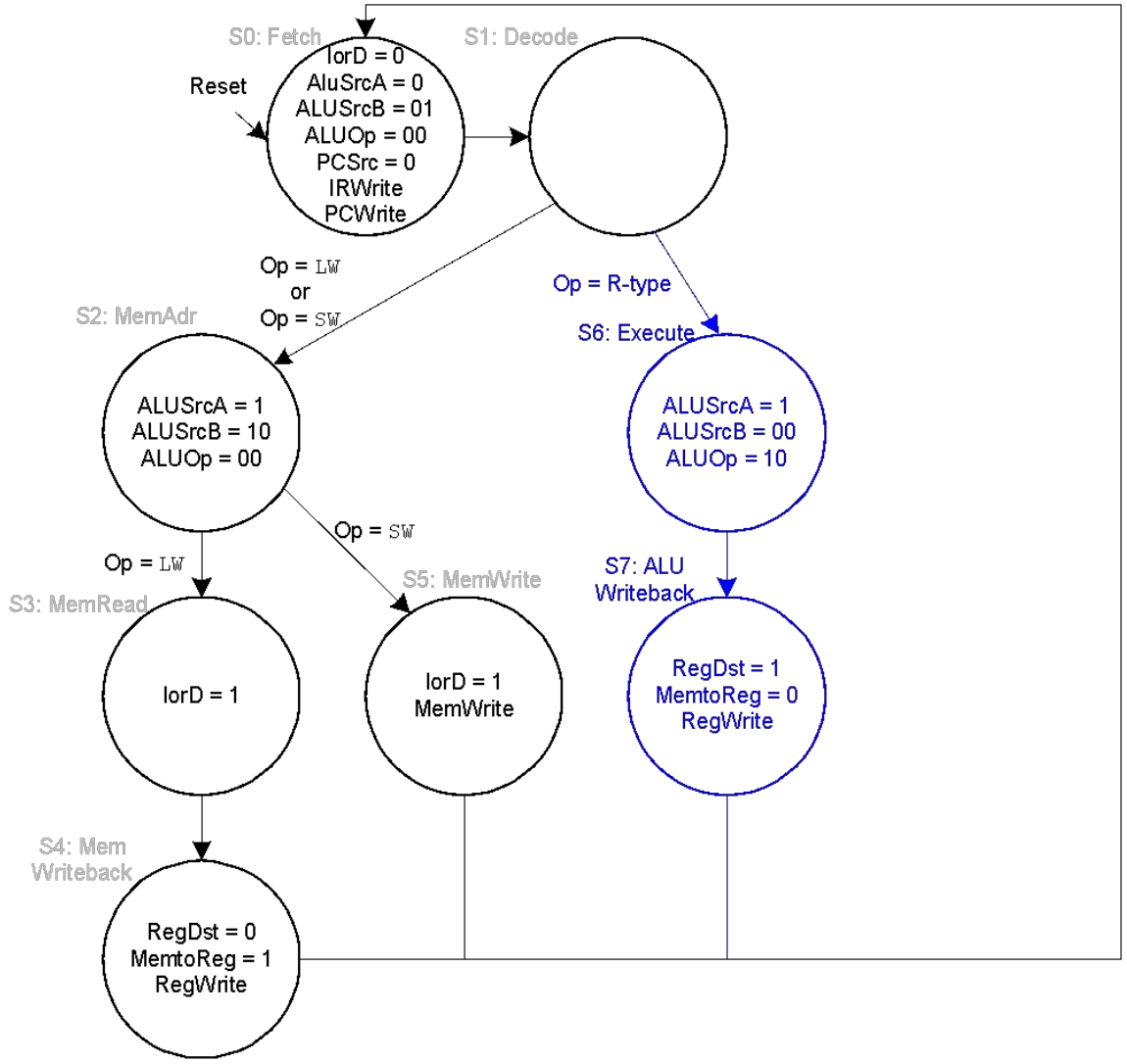


МИКРОАРХИТЕКТУРА

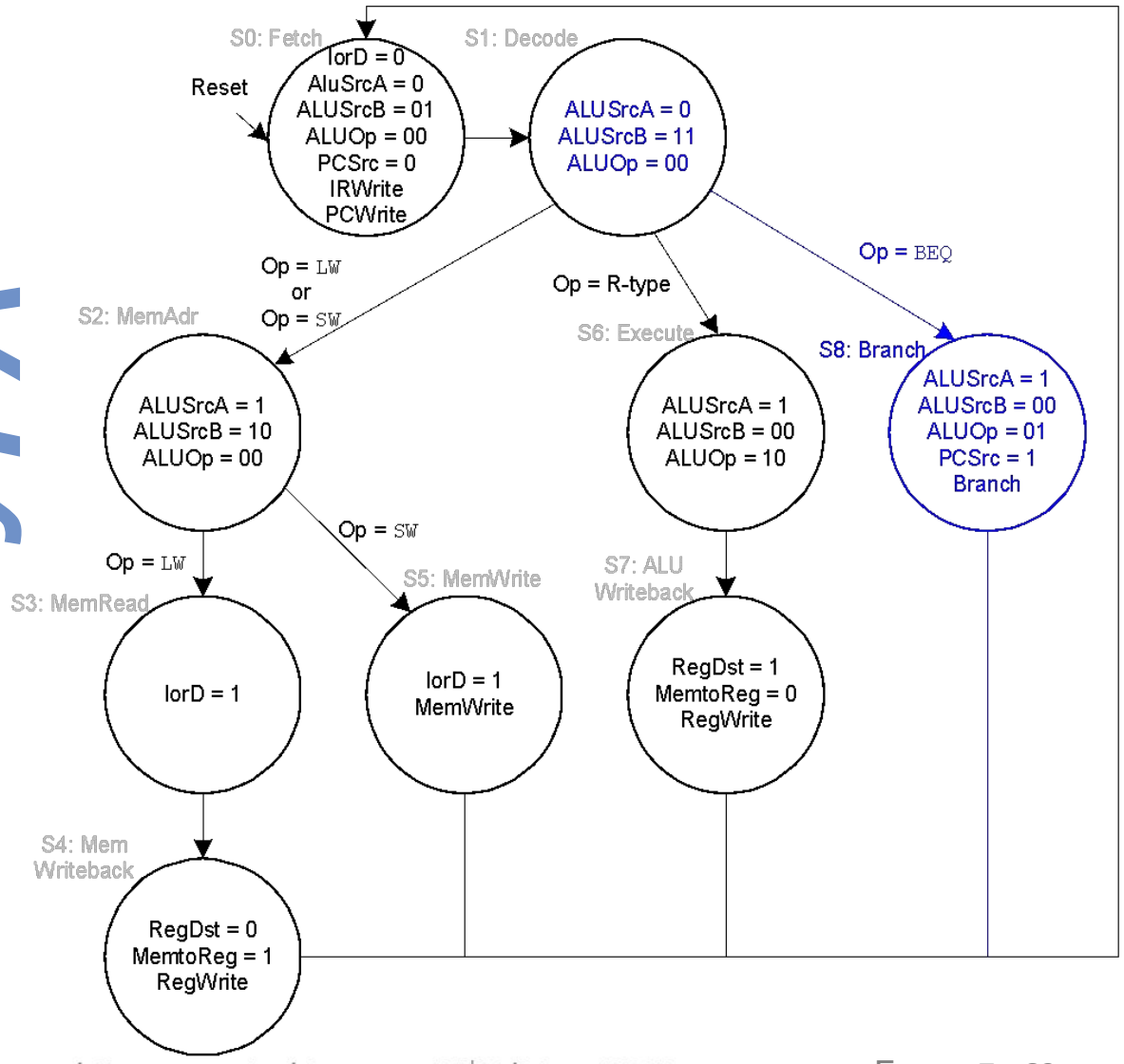
Основной управляющий автомат:



Основной управляющий автомат: R-

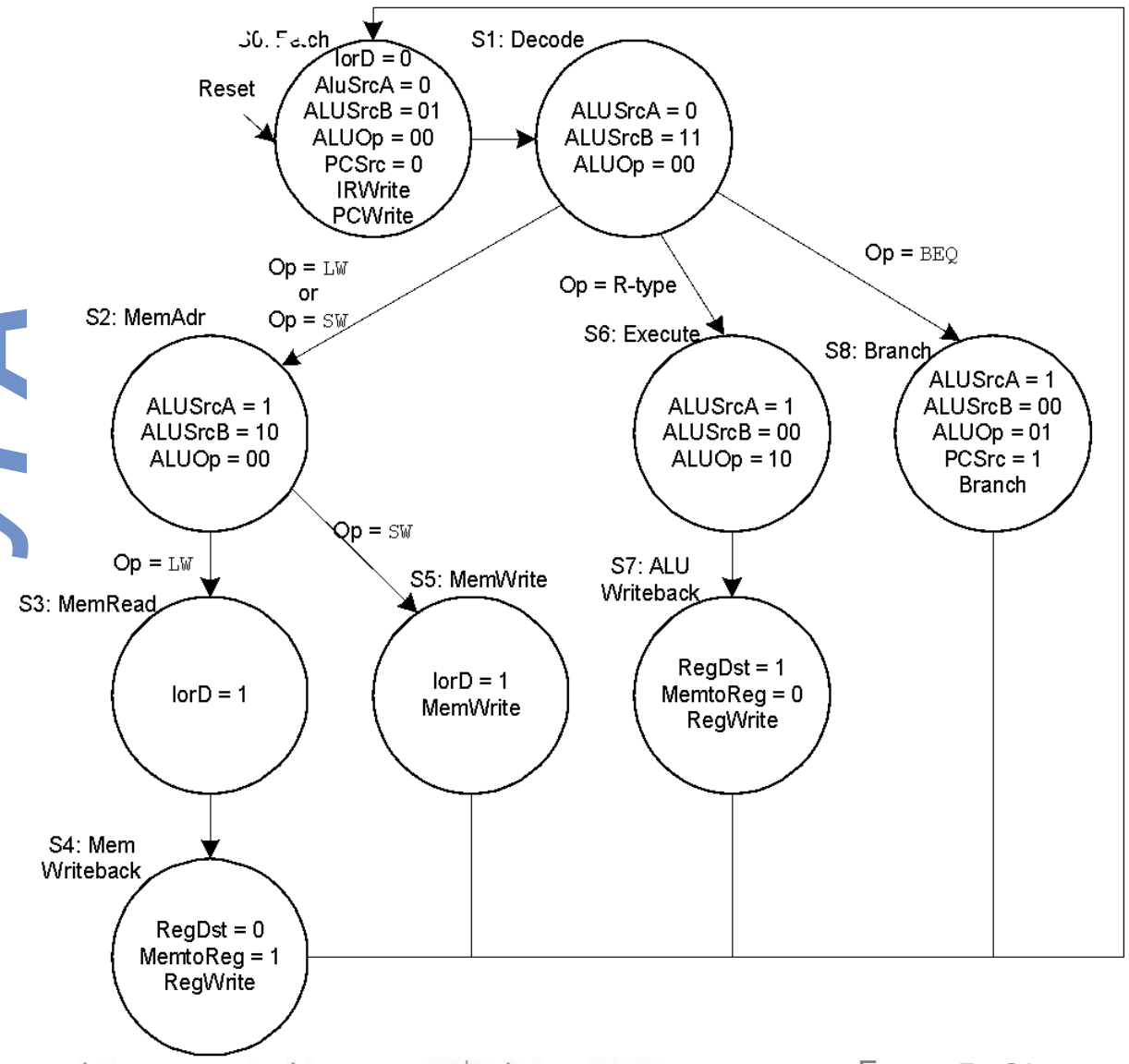


Основной управляющий автомат :



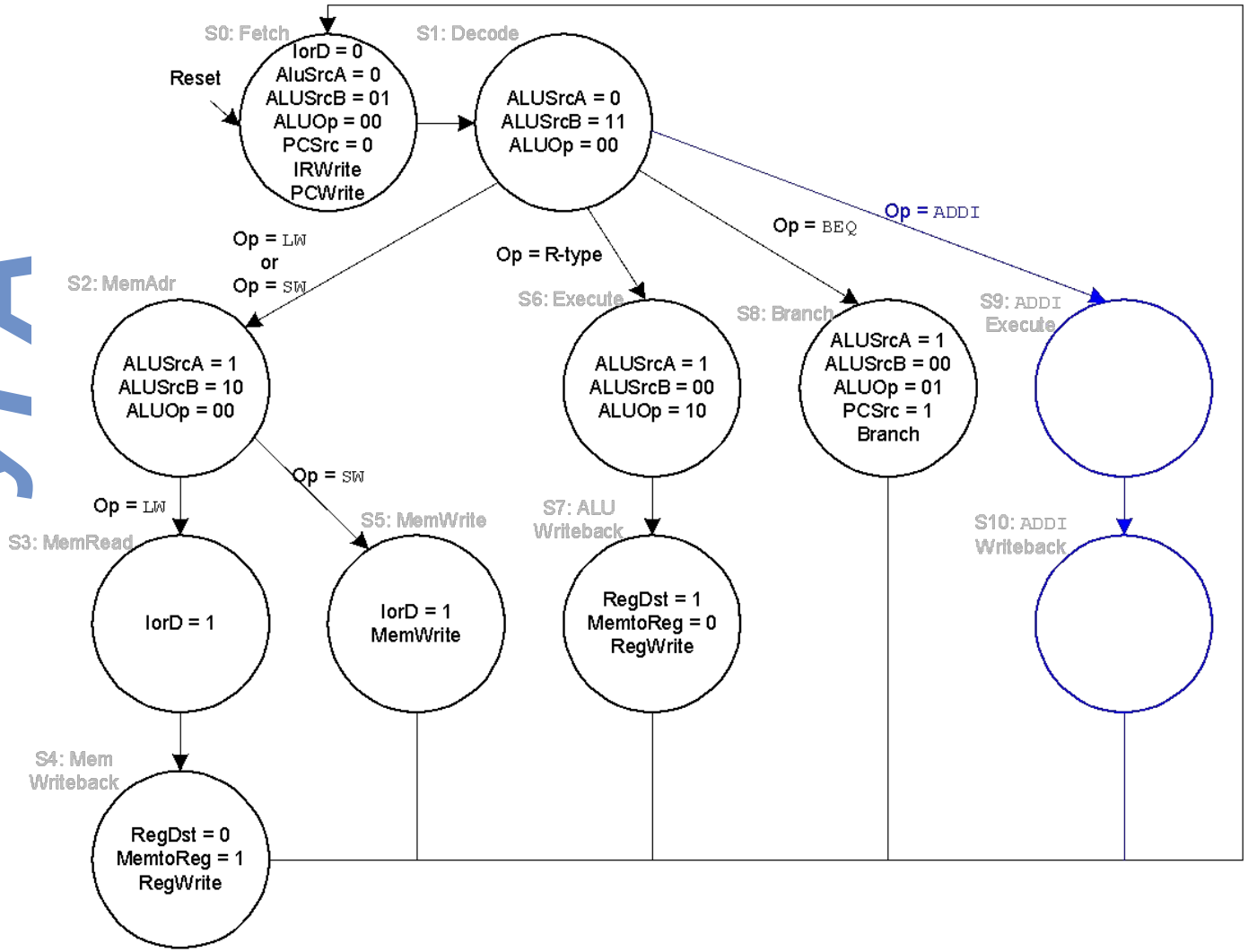
МИКРОАРХИТЕКТУРА

Основной управляющий



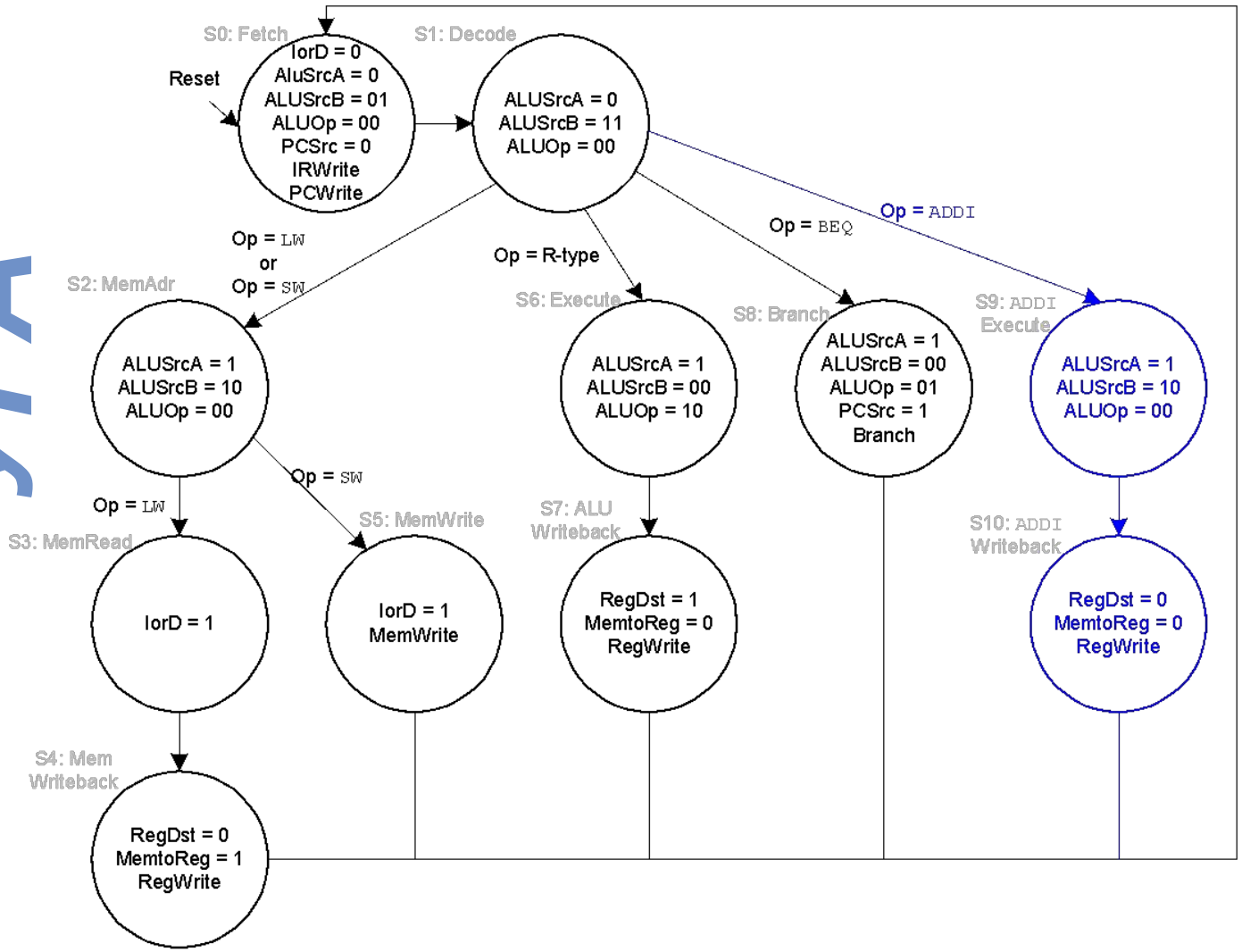
МИКРОАРХИТЕКТУРА

Добавим функционала: addi

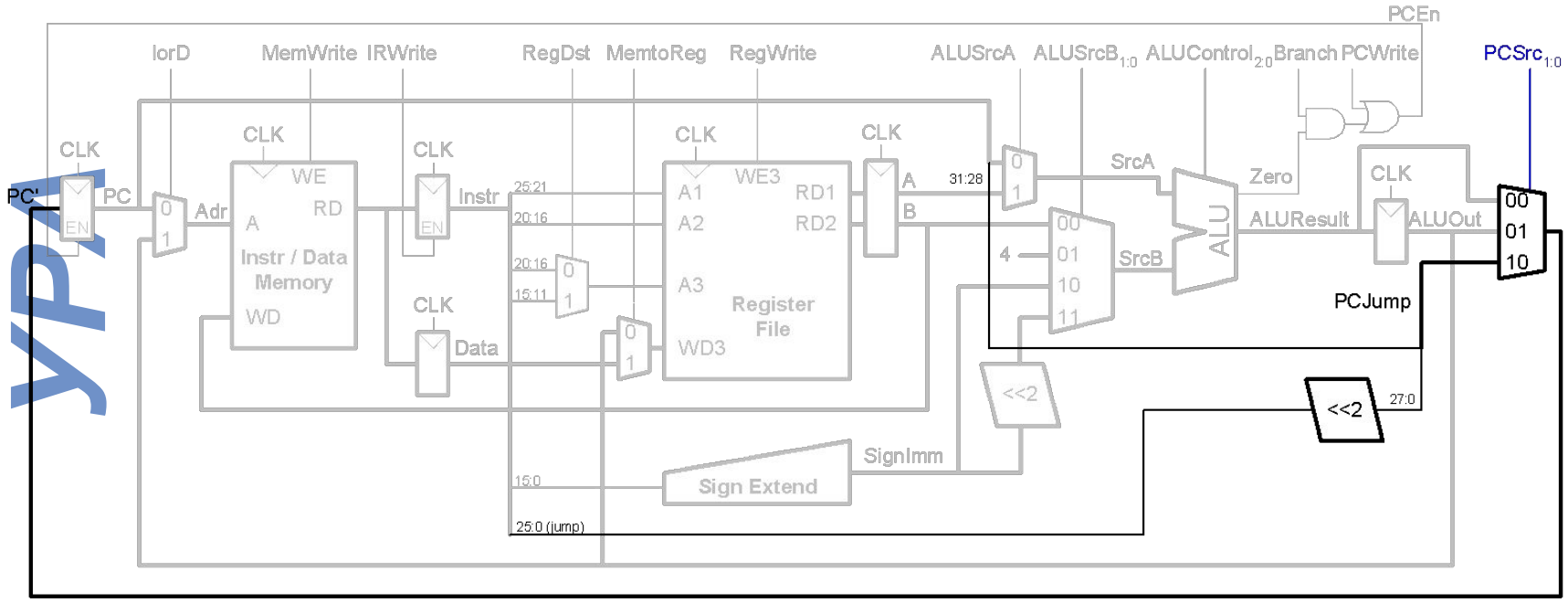


МИКРОАРХИТЕКТУРА

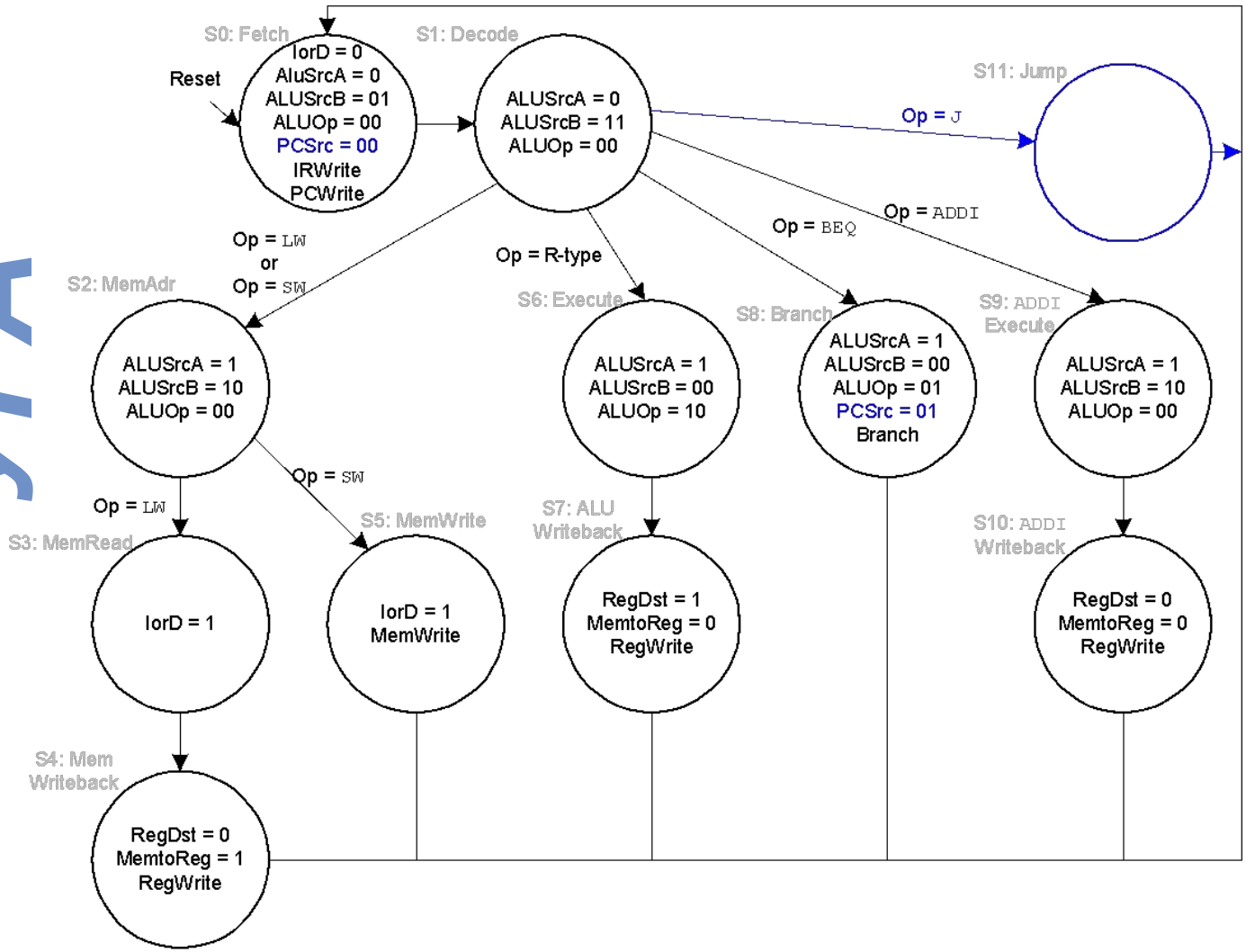
Основной управляющий автомат:



Добавим функционала: j

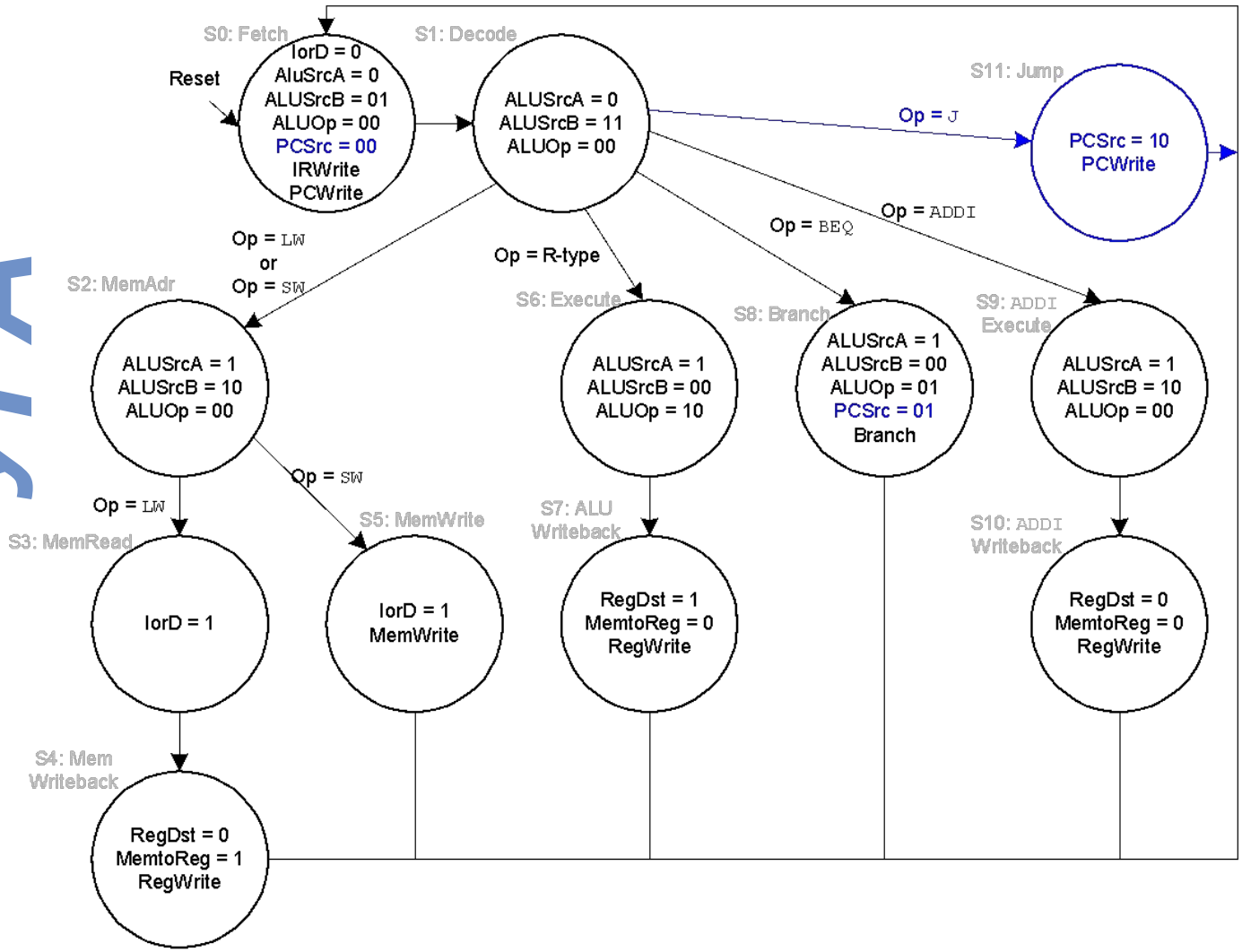


Основной управляющий автомат



Основной управляющий автомат

МИКРОАРХИТЕКТУРА



Производительность многотактного процессора

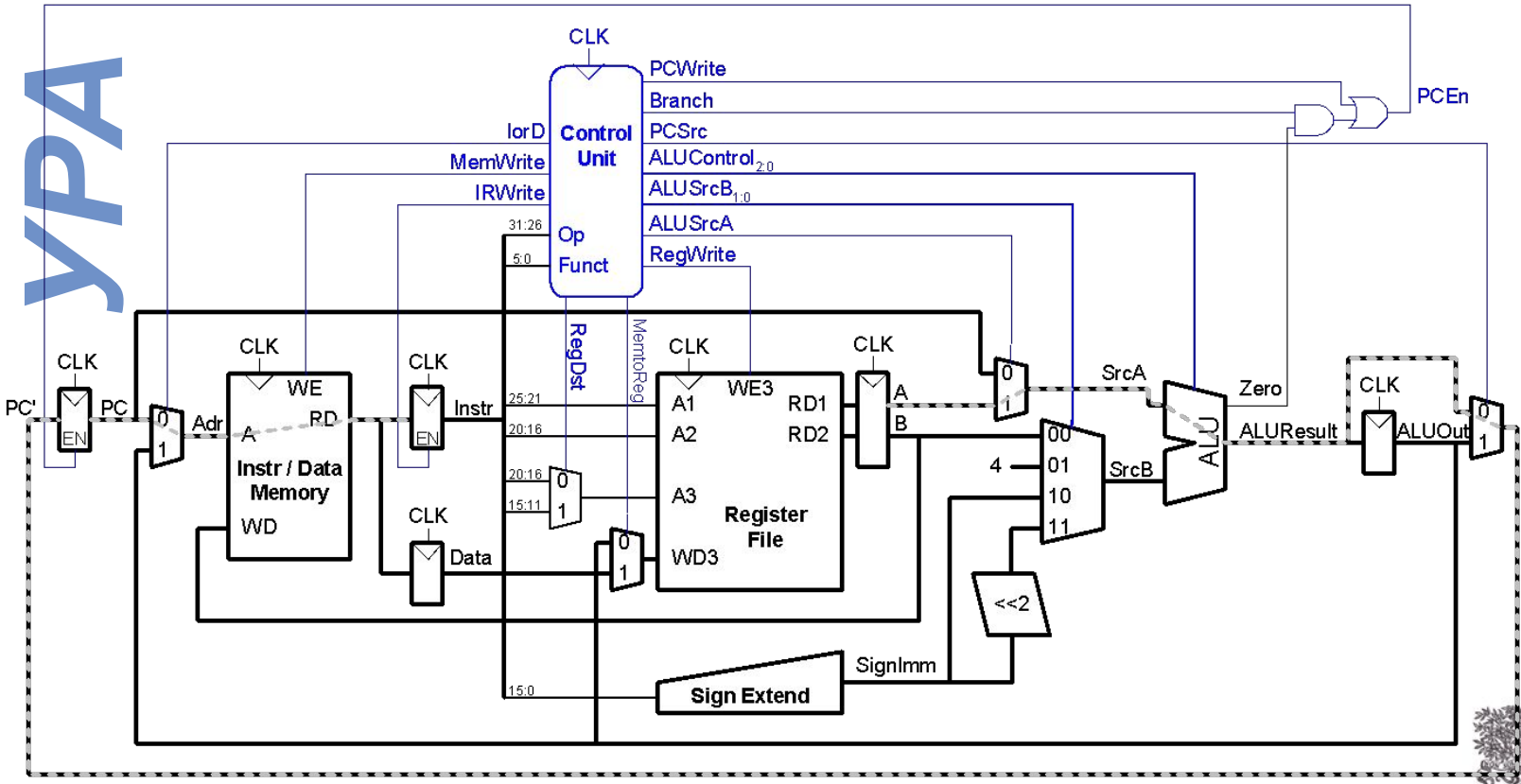
- Инструкции выполняются за разное количество тактов:
  - 3 такта: `beq`, `j`
  - 4 такта: R-тип, `sw`, `addi`
  - 5 тактов: `lw`
- CPI будет средним значением
- Тестовый набор SPECINT2000 содержит:
  - 25% инструкций `lw`
  - 10% инструкций `sw`
  - 11% условных переходов
  - 2% безусловных переходов
  - 52% инструкций R-типа

Средний CPI = $(0.11 + 0.2)(3) + (0.52 + 0.10)(4) + (0.25)(5) = \mathbf{4.12}$

Производительность многотактного процессора

Задержка самой длинной цепи комбинационной логики многотактного процессора:

$$T_c = t_{pcq} + t_{\text{mux}} + \max(t_{\text{ALU}} + t_{\text{mux}}, t_{\text{mem}}) + t_{\text{setup}}$$



Посчитаем производительность многотактного процессора

Параметр	Обозначение	Задержка (пс)
Время записи в регистр	t_{pcq_PC}	30
Время предустановки регистра	t_{setup}	20
Задержка мультиплексора	t_{mux}	25
Задержка АЛУ	t_{ALU}	200
Задержка считывания из памяти	t_{mem}	250
Задержка считывания из регистрового файла	t_{RFread}	150
Время предустановки регистрового файла	$t_{RFsetup}$	20

$$T_c = ?$$

Посчитаем производительность многотактного процессора

$$
\begin{aligned}
T_c &= t_{pcq\_PC} + t_{\text{mux}} + \max(t_{\text{ALU}} + t_{\text{mux}}, t_{\text{mem}}) + t_{\text{setup}} \\
&= t_{pcq\_PC} + t_{\text{mux}} + t_{\text{mem}} + t_{\text{setup}} \\
&= [30 + 25 + 250 + 20] \text{ пс} \\
&= \mathbf{325 \text{ пс}}
\end{aligned}
$$

Посчитаем производительность многотактного процессора

Предположим, в программе 100 миллиардов инструкций:

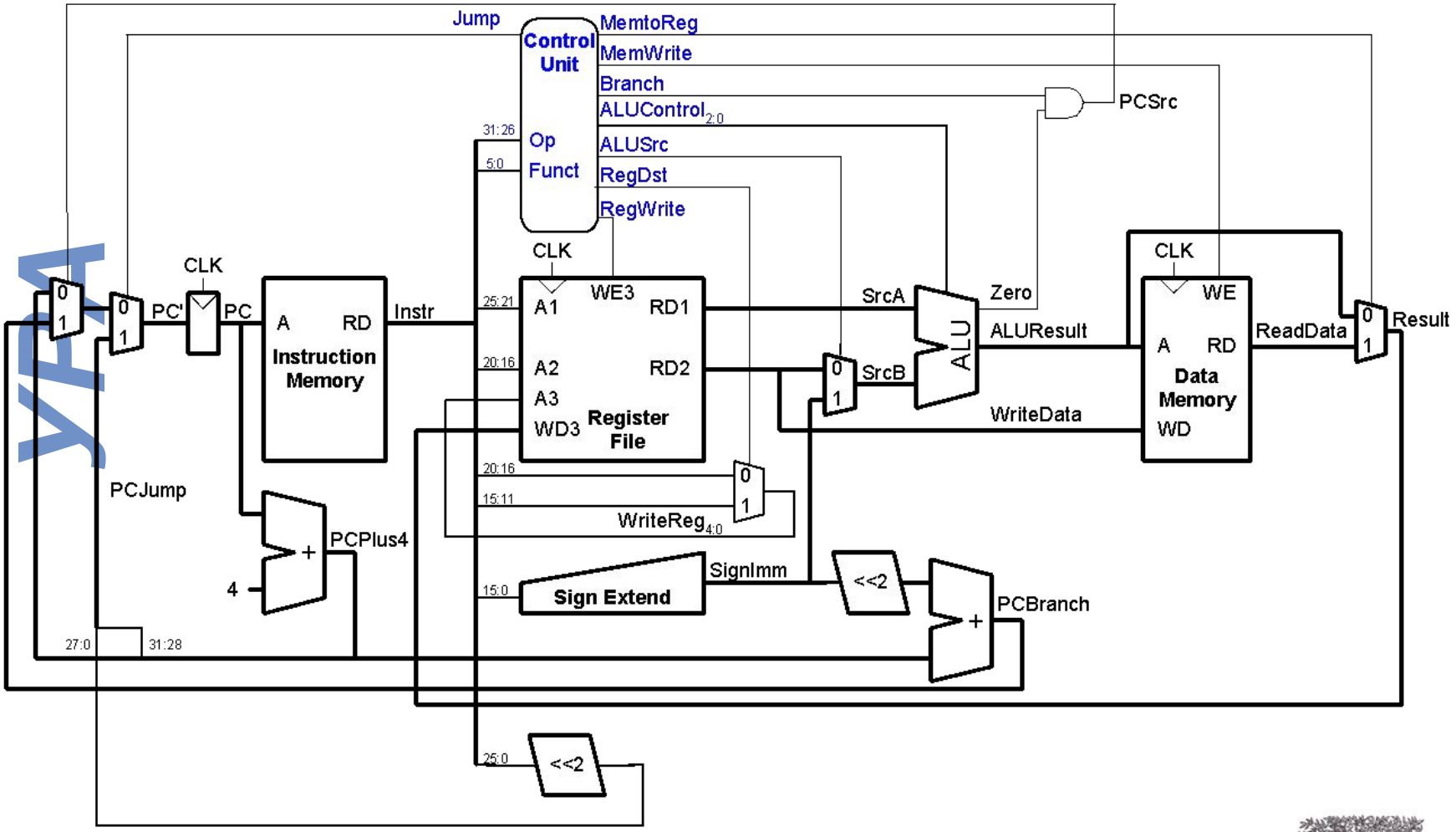
- CPI = 4.12
- T_c = 325 пс

$$\text{Время выполнения} = (\#\text{ инструкции}) \times \text{CPI} \times T_c$$
$$= (100 \times 10^9)(4.12)(325 \times 10^{-12})$$
$$= \mathbf{133.9\ секунд}$$

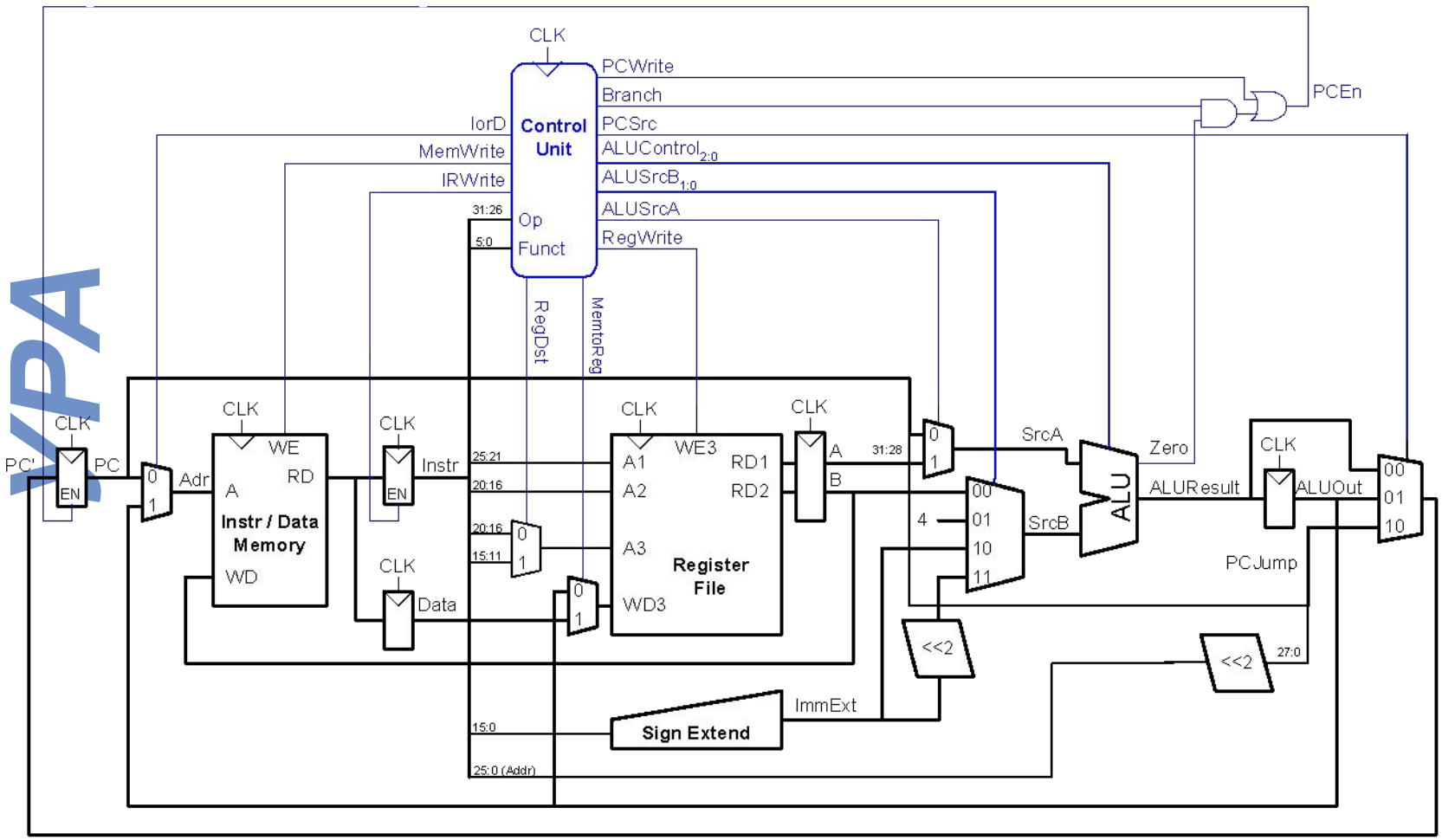
Это **больше**, чем для однотактного процессора (92.5 секунд). Почему?

- У разных команд разная длительность выполнения
- Дополнительные задержки на каждом шаге (t_{pcq} + t_{setup} = 50 пс)

Повторение: однотактный процессор



Повторение: многотактный процессор

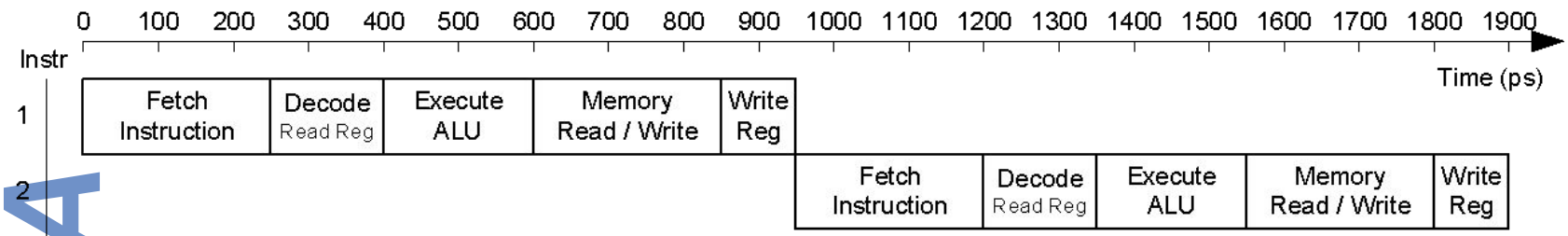


Конвейерный MIPS процессор

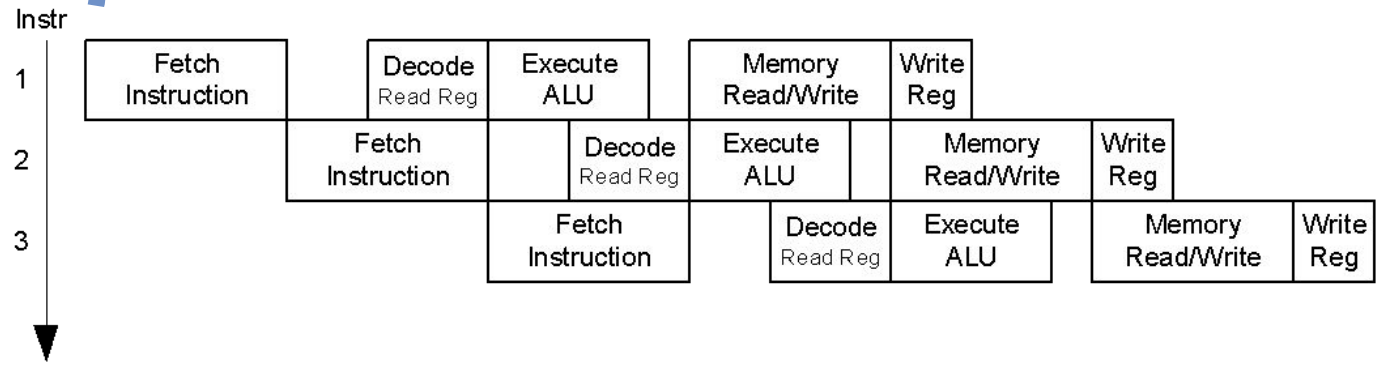
- Временной параллелизм
- Разделим однотактный процессор на 5 стадий:
  - Выборка
  - Декодирование
  - Выполнение
  - Доступ к памяти
  - Запись результатов
- Добавим регистры между стадиями конвейера

Однотактный / Конвейерный

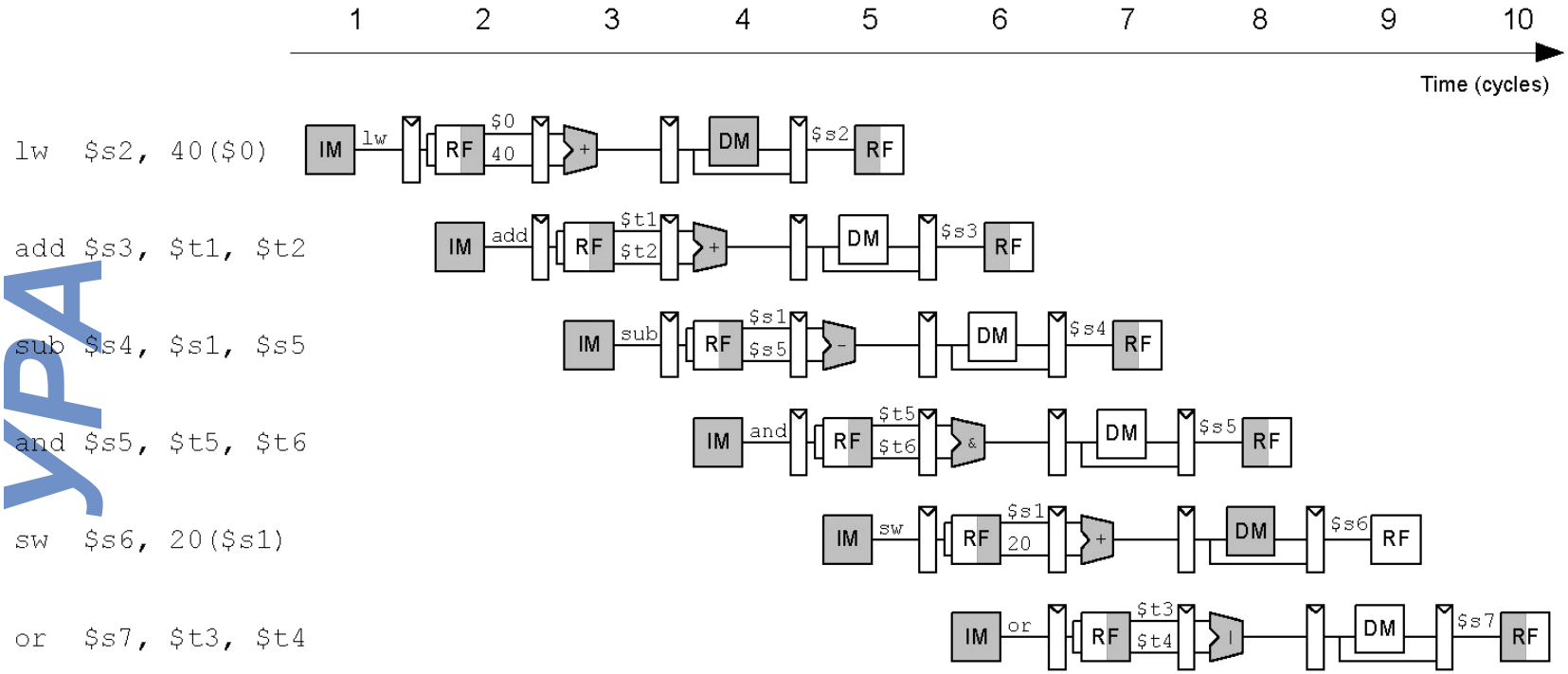
Single-Cycle



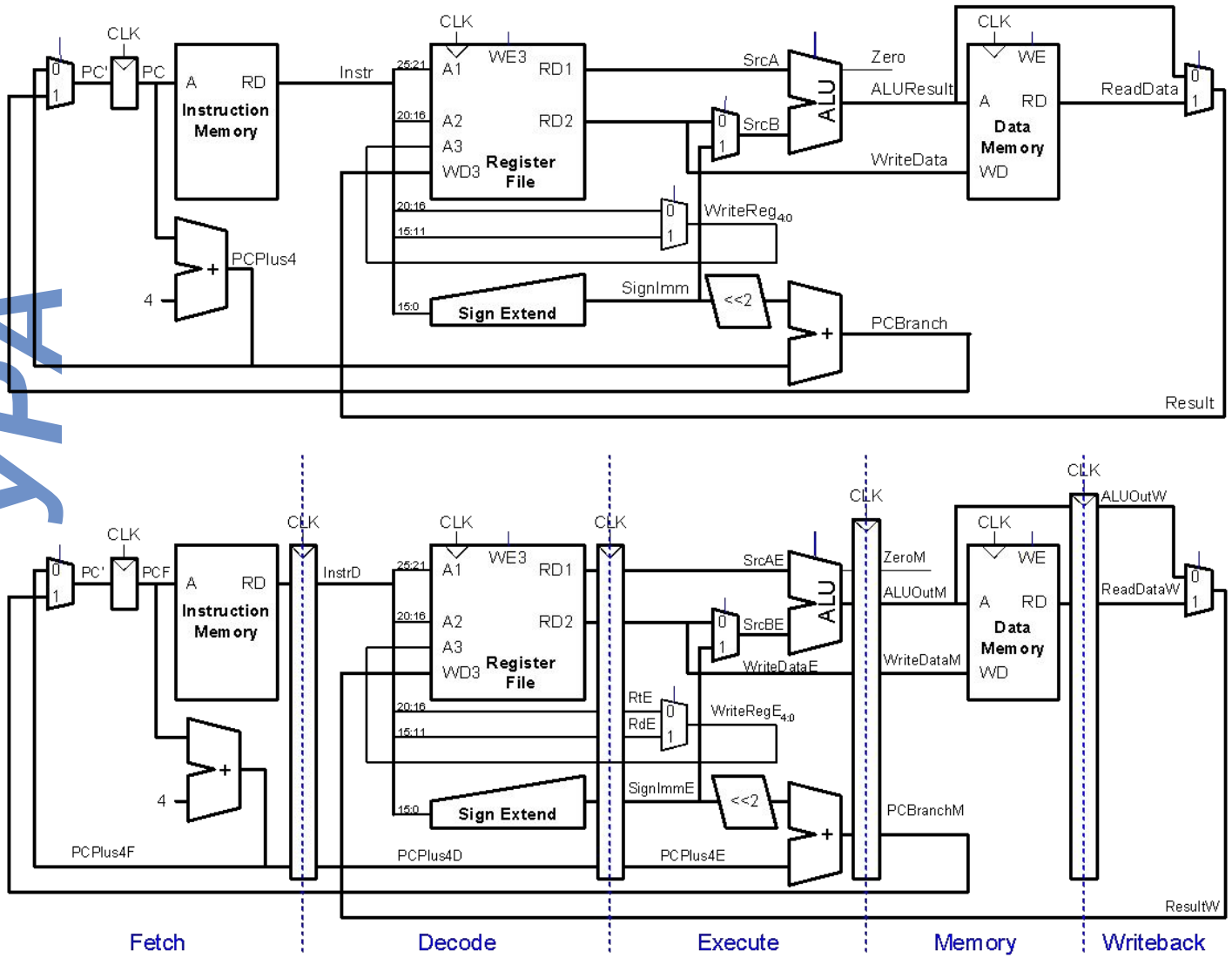
Pipelined



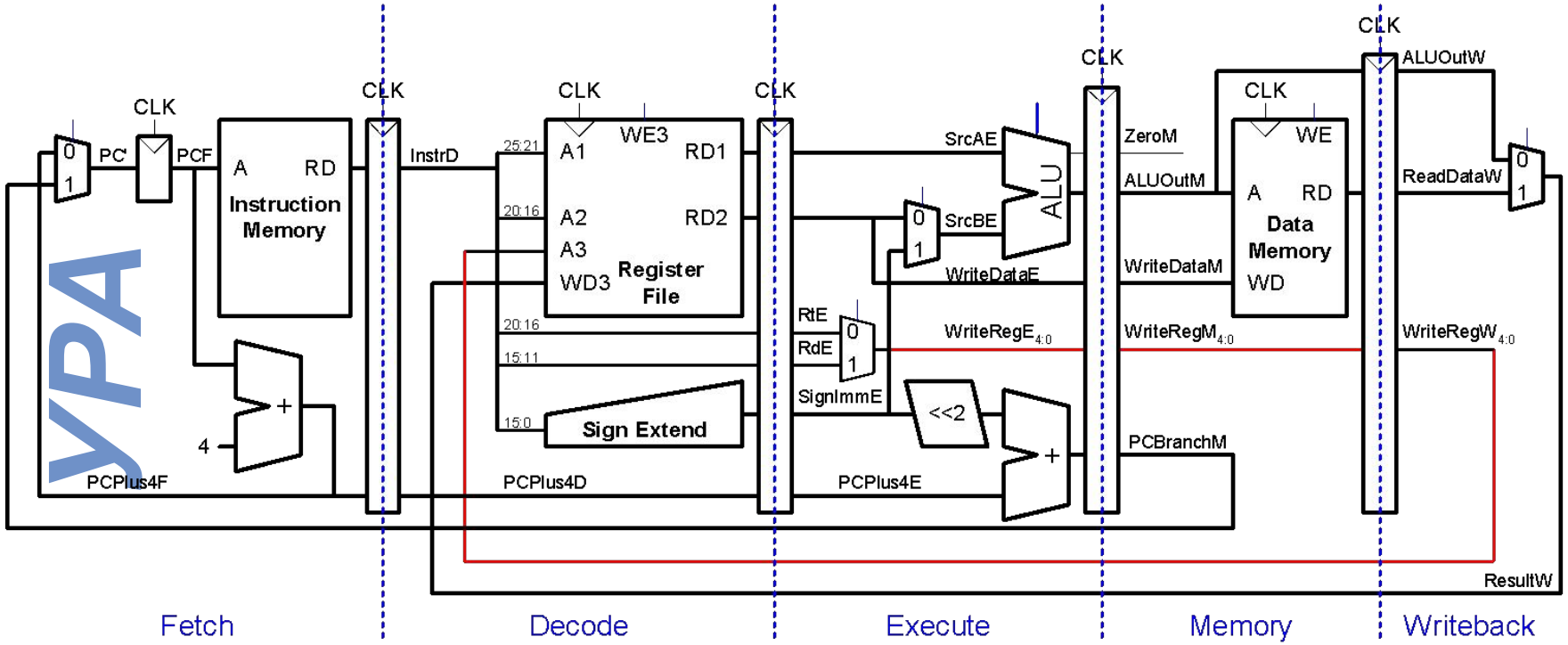
Абстрактное представление конвейера



Однотактный и конвейерный тракт данных



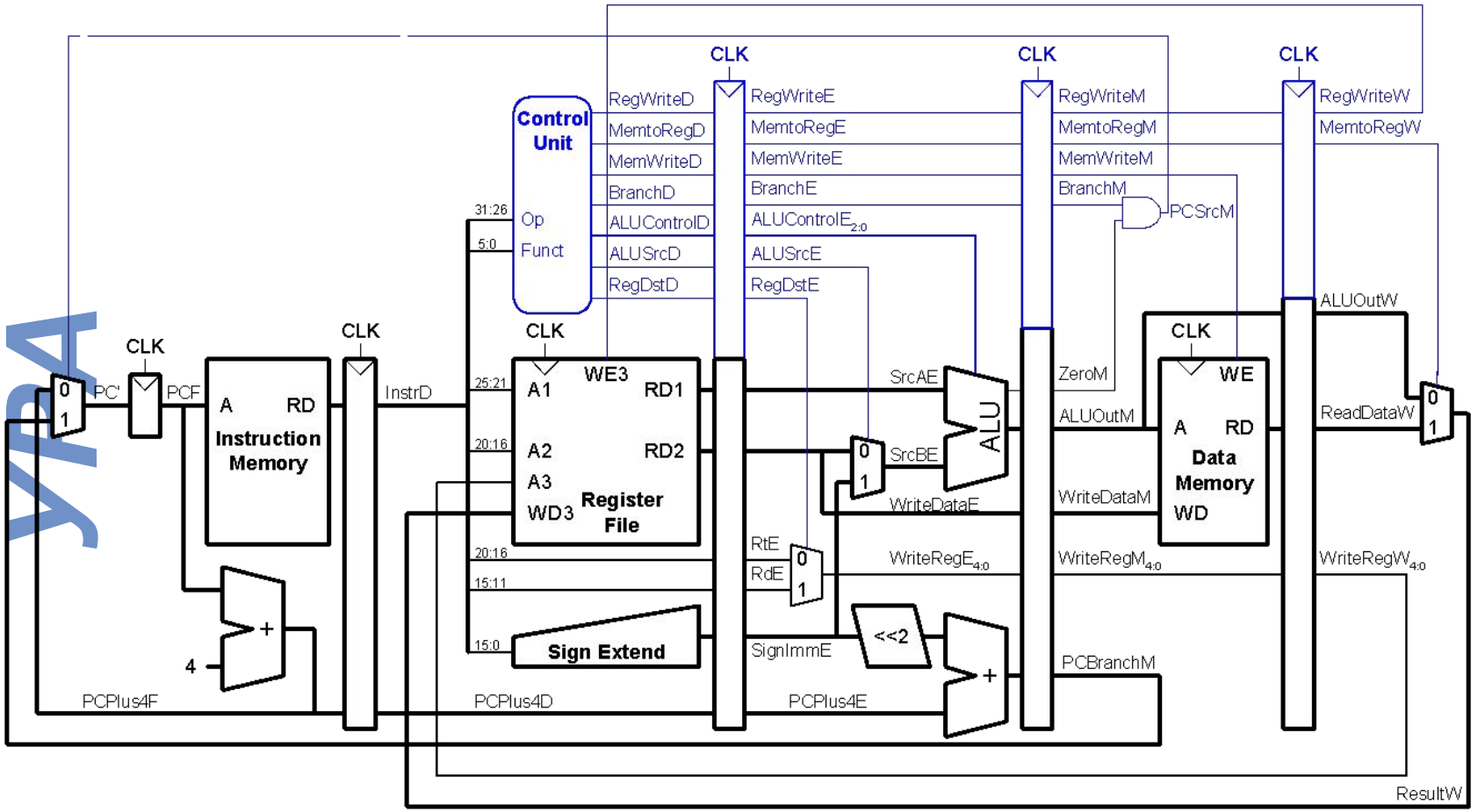
Исправленный конвейерный тракт данных



Теперь *WriteRegW* и *ResultW* подаются на входы регистрового файла в стадии Writeback одновременно.

МИКРОАРХИТЕКТ УРА

Управление конвейерным процессором

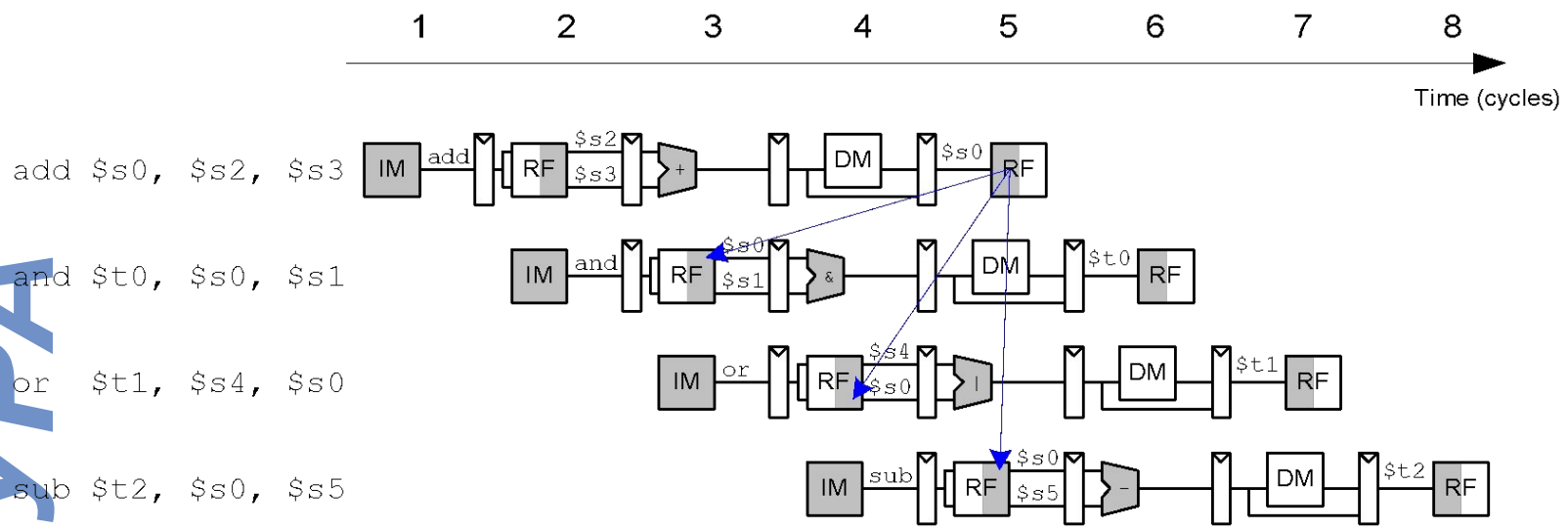


- То же устройство управления, что и в однотактном процессоре
- Сигналы управления доходят до соответствующей стадии с задержкой (сигналы управления тоже конвейеризируются)

Конфликты конвейера

- В конвейере выполняется несколько инструкций одновременно
- Конфликты случаются когда одна инструкция зависит от результата другой, еще не завершенной инструкции
- Типы конфликтов:
  - **Конфликты данных:** результат инструкции еще не записан в регистр, а следующая инструкция уже пытается считать этот регистр
  - **Конфликты управления:** процессор выбирает из памяти следующую инструкцию до того, как стало ясно, какую именно инструкцию надо выбрать(возникают из-за условных переходов)

Конфликты данных

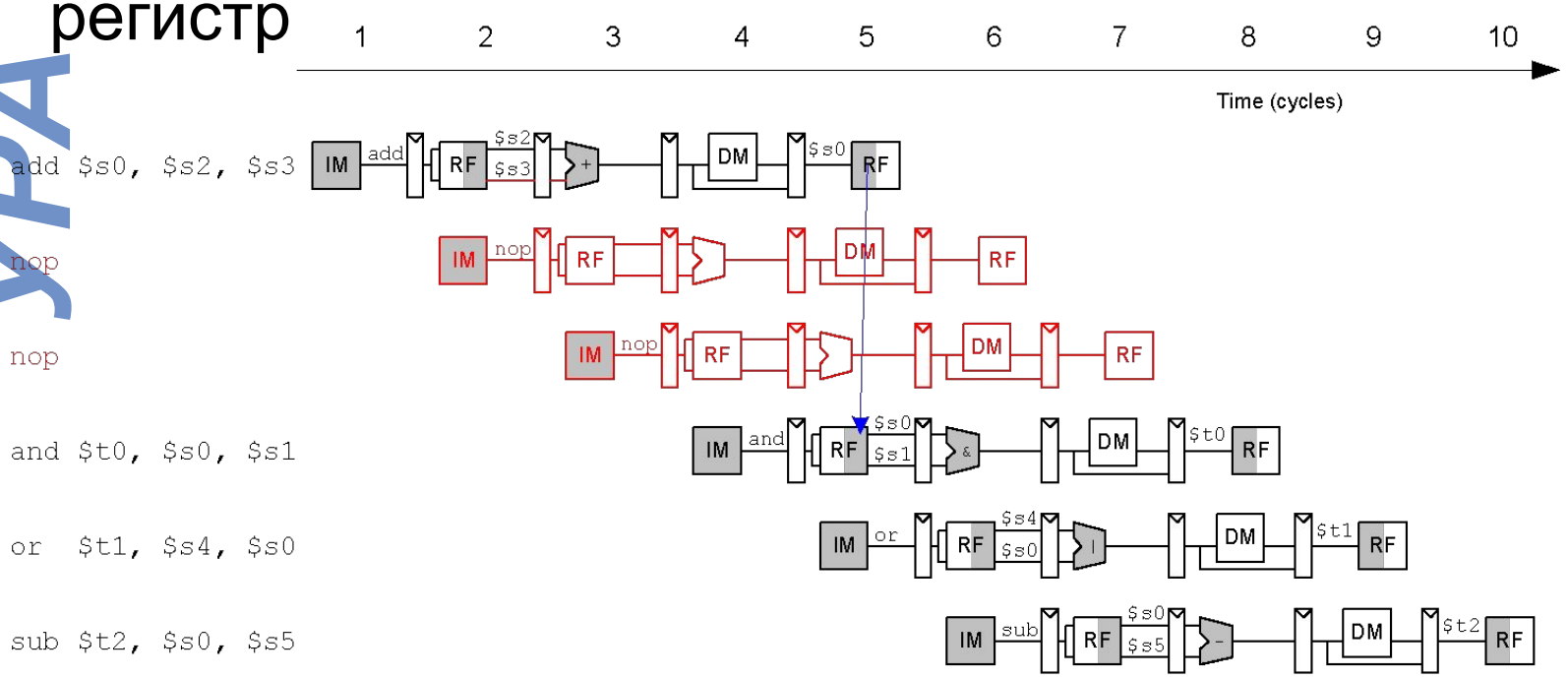


Разрешение конфликтов

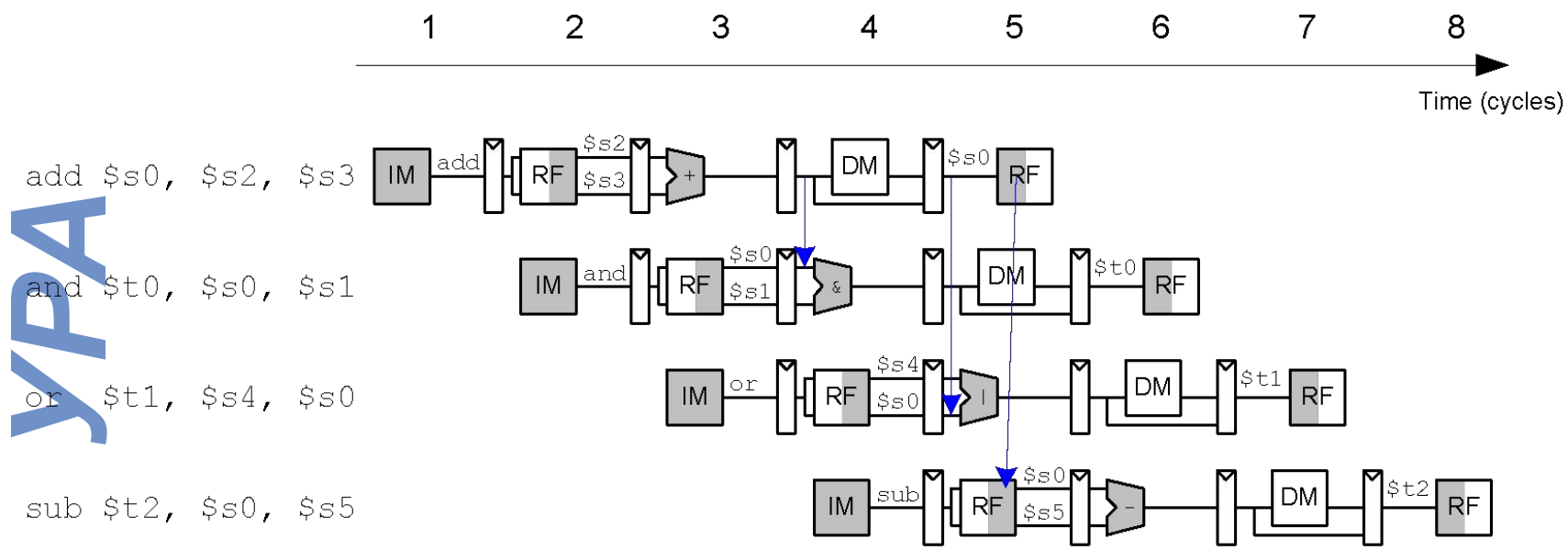
- Можно вставлять пустые инструкции (`nop`) в код программы перед компиляцией или во время компиляции
- Во время выполнения программы реализовать аппаратную передачу данных с одного этапа конвейера на другой не дожидаясь завершения инструкции
- Во время выполнения программы останавливать (stall) некоторые этапы конвейера до тех пор, пока проблемная инструкция не запишет в регистровый файл результат, от которого зависят инструкции на остановленных этапах

Устранение конфликтов на уровне компилятора

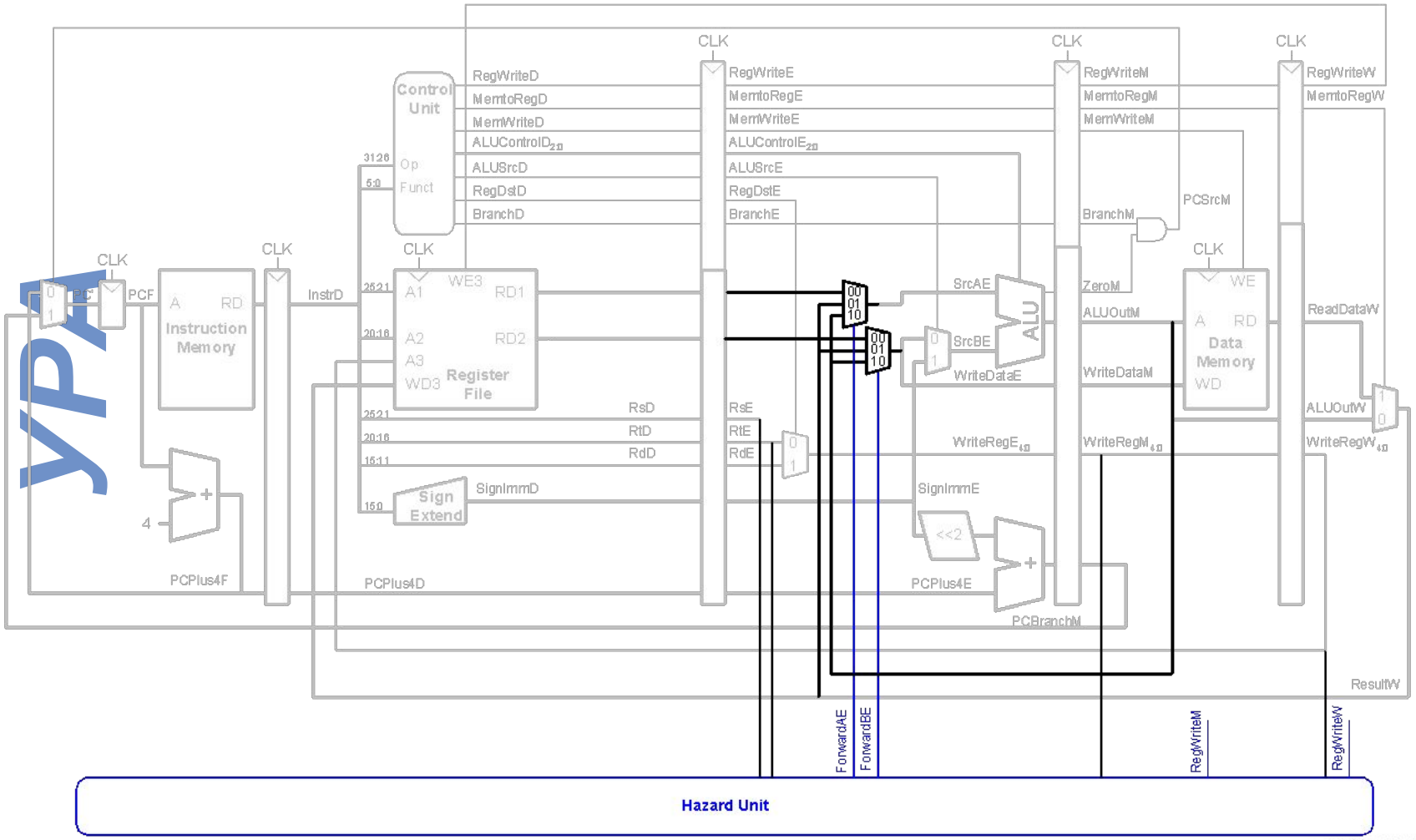
- Вставить в код достаточно пустых инструкций (`nop`), которые будут заполнять стадии конвейера, пока необходимый результат не будет записан в регистр



Передача данных между стадиями (Forwarding, Bypass)



Передача данных между стадиями (Forwarding, Bypass)



Передача данных между стадиями (Forwarding, Bypass)

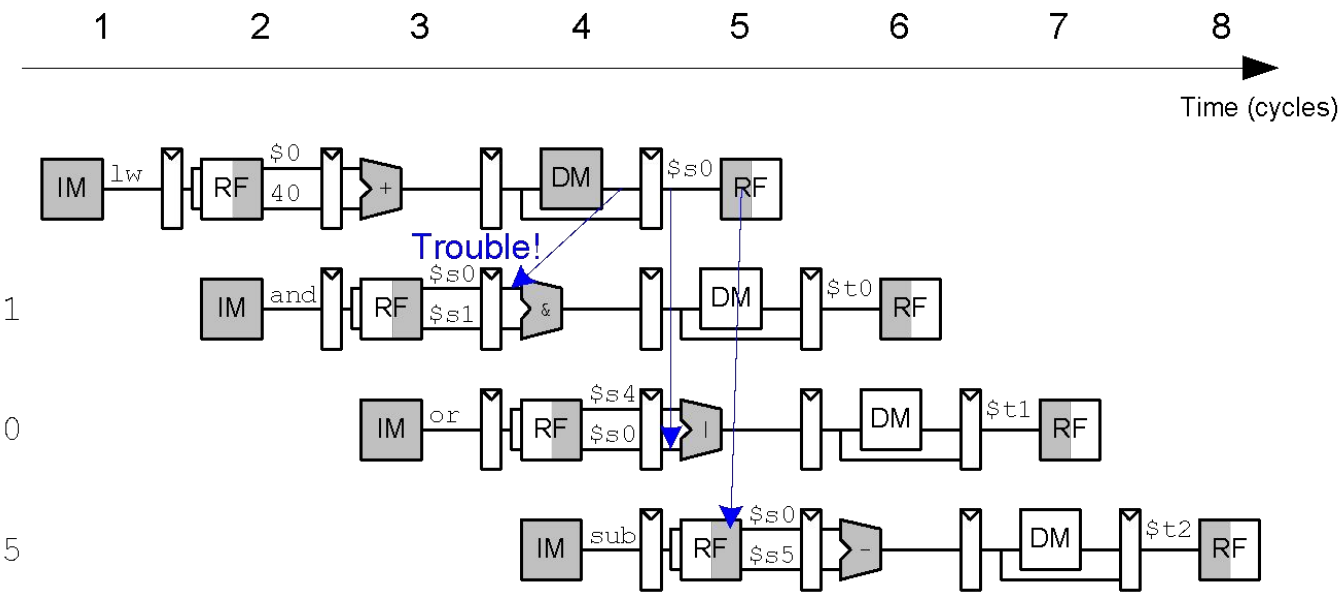
- Можно передавать необходимые данные на этап Выполнения с этапов:
  - Доступа к памяти или
  - Записи результатов в регистровый файл
- Управляющая логика для *ForwardAE*:

```
if      ((rsE != 0) AND (rsE == WriteRegM) AND RegWriteM)
 then   ForwardAE = 10
else if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW)
 then   ForwardAE = 01
else        ForwardAE = 00
```

Управляющая логика для *ForwardBE* похожа, но нужно заменить *rsE* на *rtE*

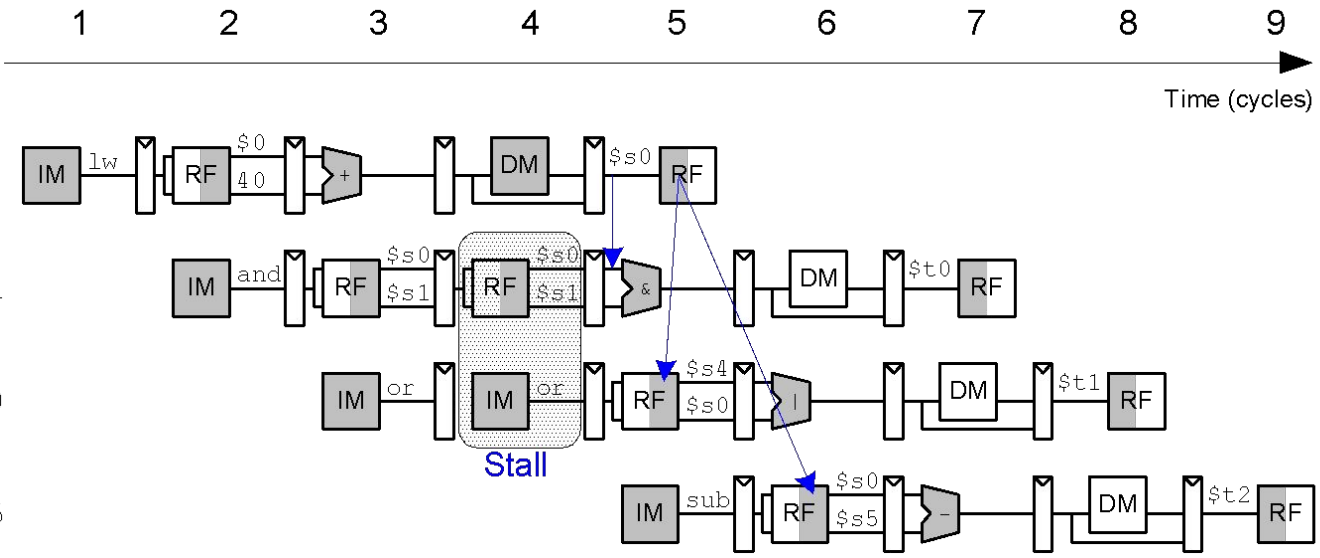
Останов конвейера

```
lw $s0, 40($0)
and $t0, $s0, $s1
or $t1, $s4, $s0
sub $t2, $s0, $s5
```

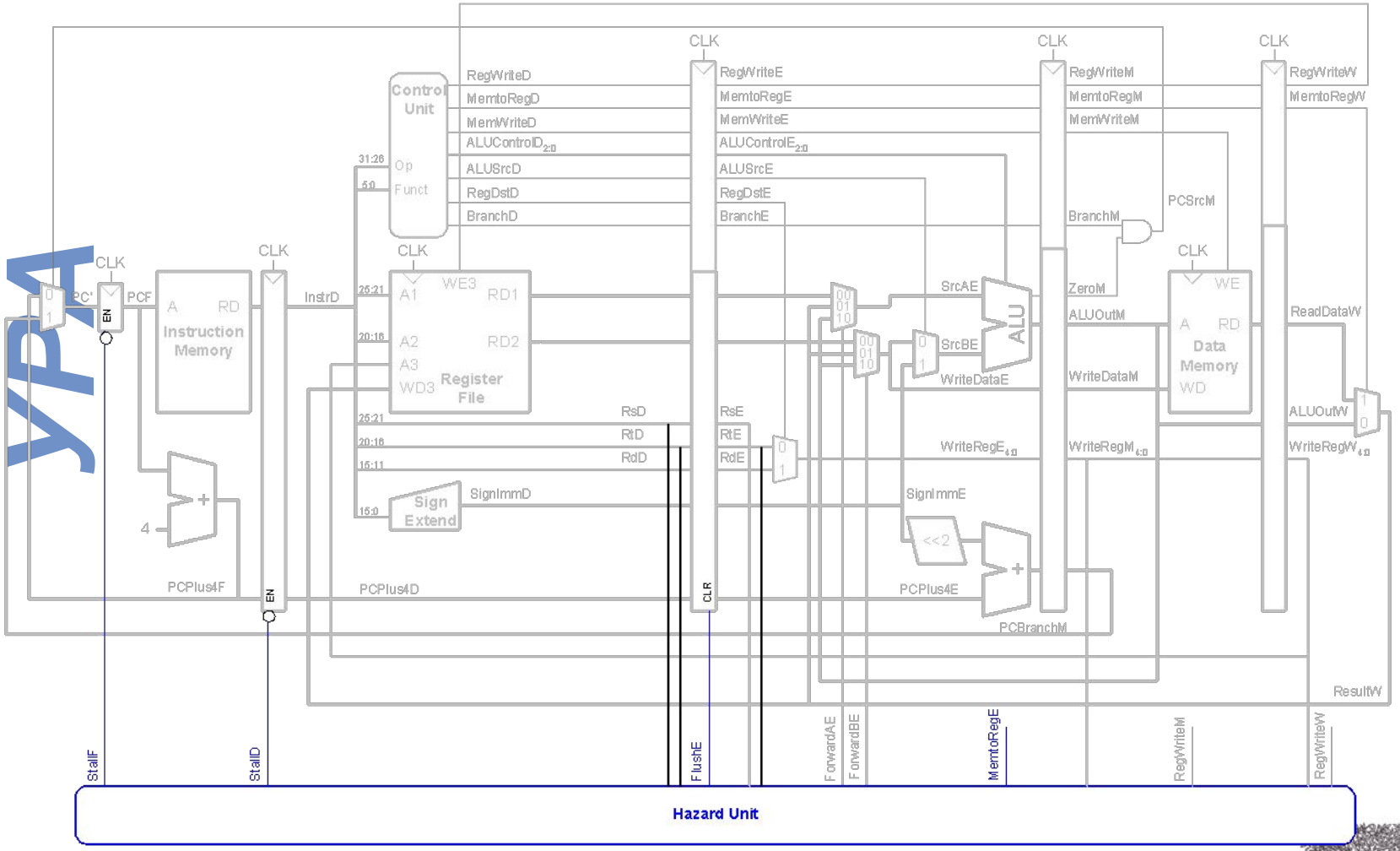


Останов конвейера

```
lw $s0, 40($0)
and $t0, $s0, $s1
or  $t1, $s4, $s0
sub $t2, $s0, $s5
```



Останов конвейера



Логика управления остановом (для инструкции `lw`)

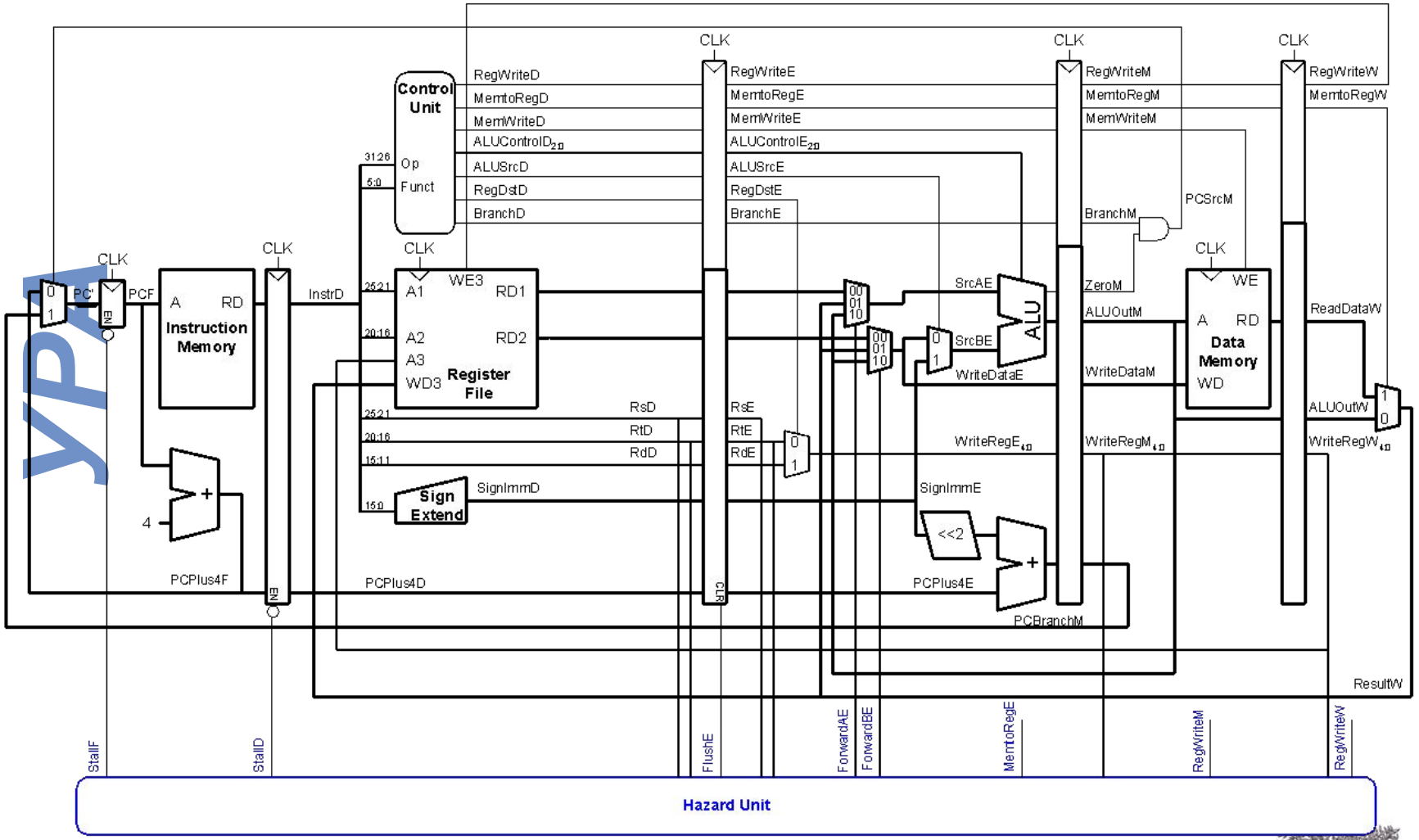
```
lwstall =
  ((rsD==rtE) OR (rtD==rtE)) AND MemtoRegE

StallF = StallD = FlushE = lwstall
```

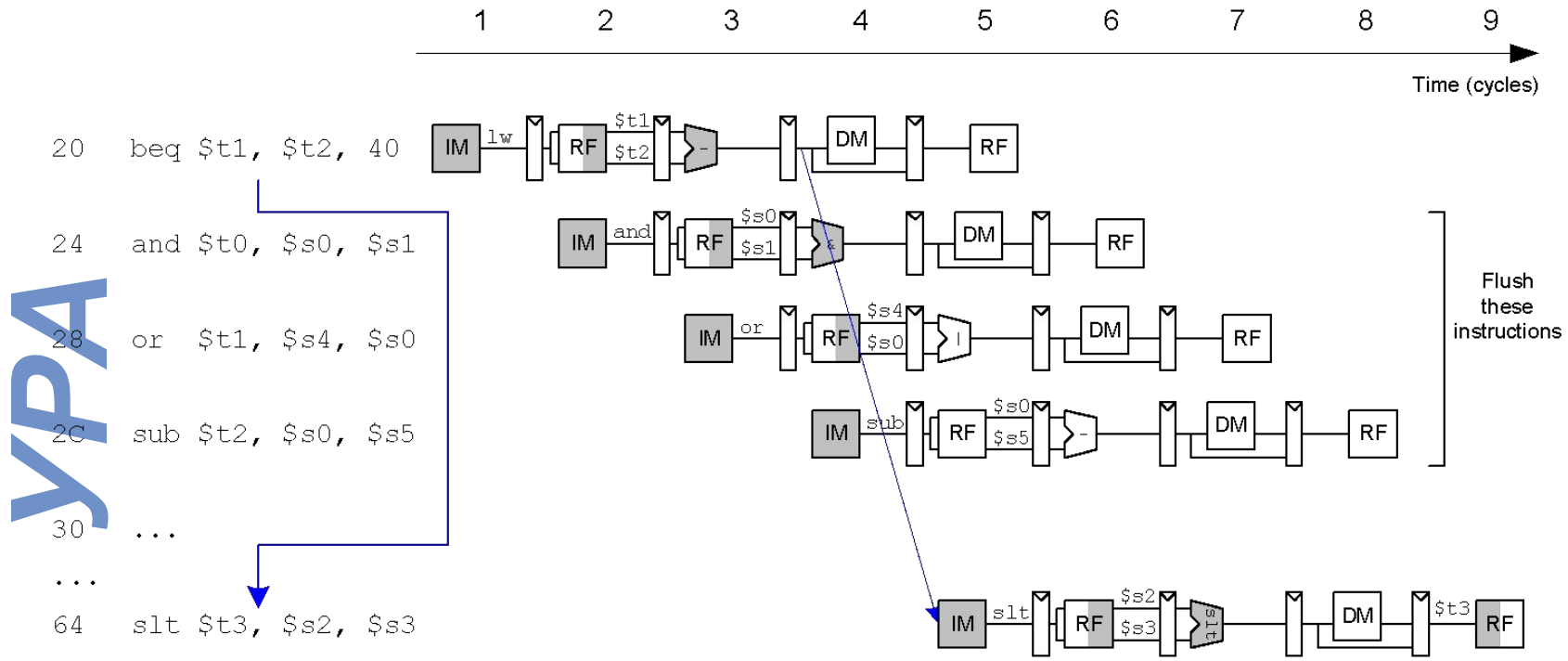
Конфликты управления

- **beq:**
  - Будет выполнен условный переход или нет становится известно только на 4-й стадии конвейера
  - Пока это не станет известно, инструкции следующие за инструкцией условного перехода продолжают попадать в конвейер
  - В случае необходимости условного перехода эти инструкции (идущие после `beq`) не должны быть выполнены и их необходимо удалить из конвейера
- **Цена неправильного предсказания результата условного перехода**
  - Количество инструкций, которые необходимо удалить из конвейера, если переход все таки произойдет
  - Это количество можно уменьшить, проверяя

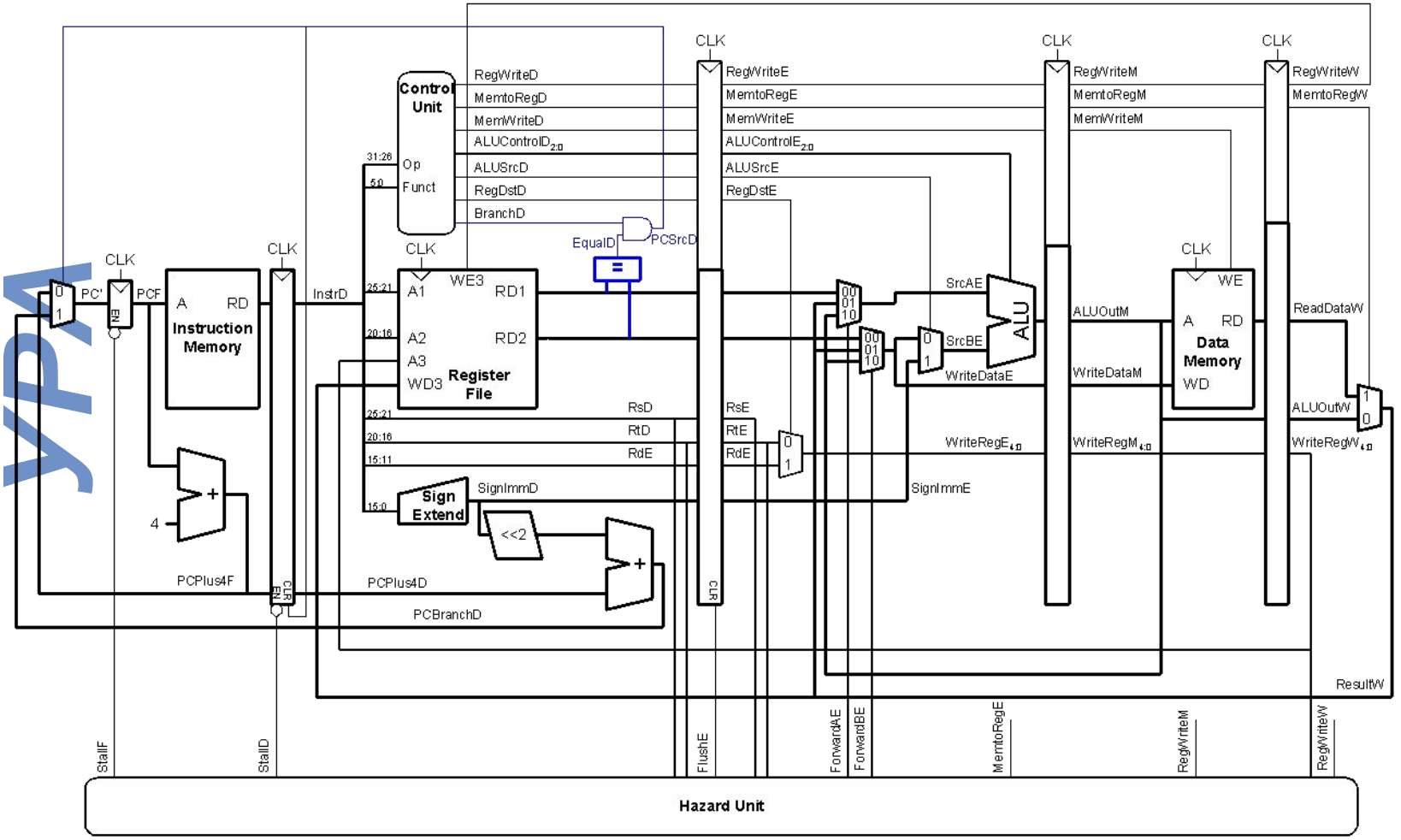
Конвейер до разрешения конфликтов управления



Конфликты управления

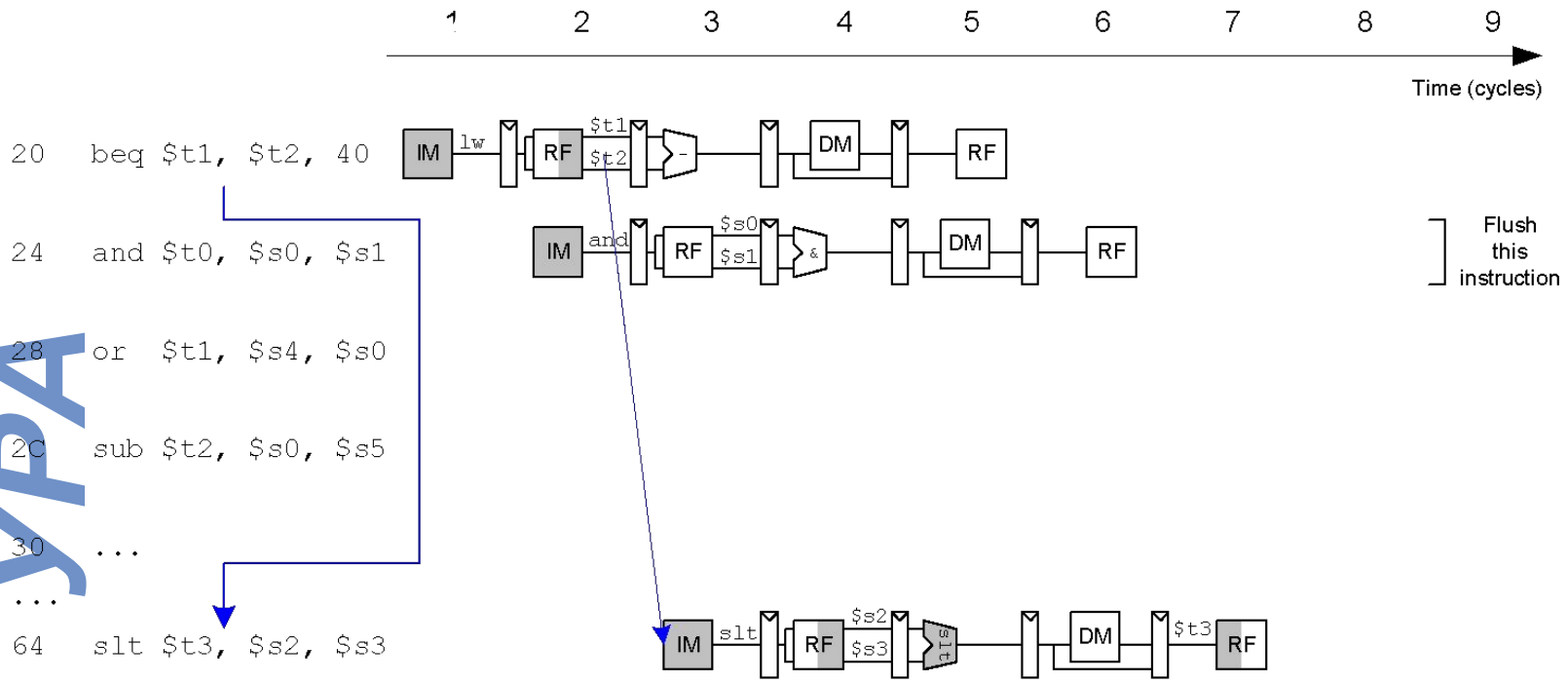


Ранняя проверка условия



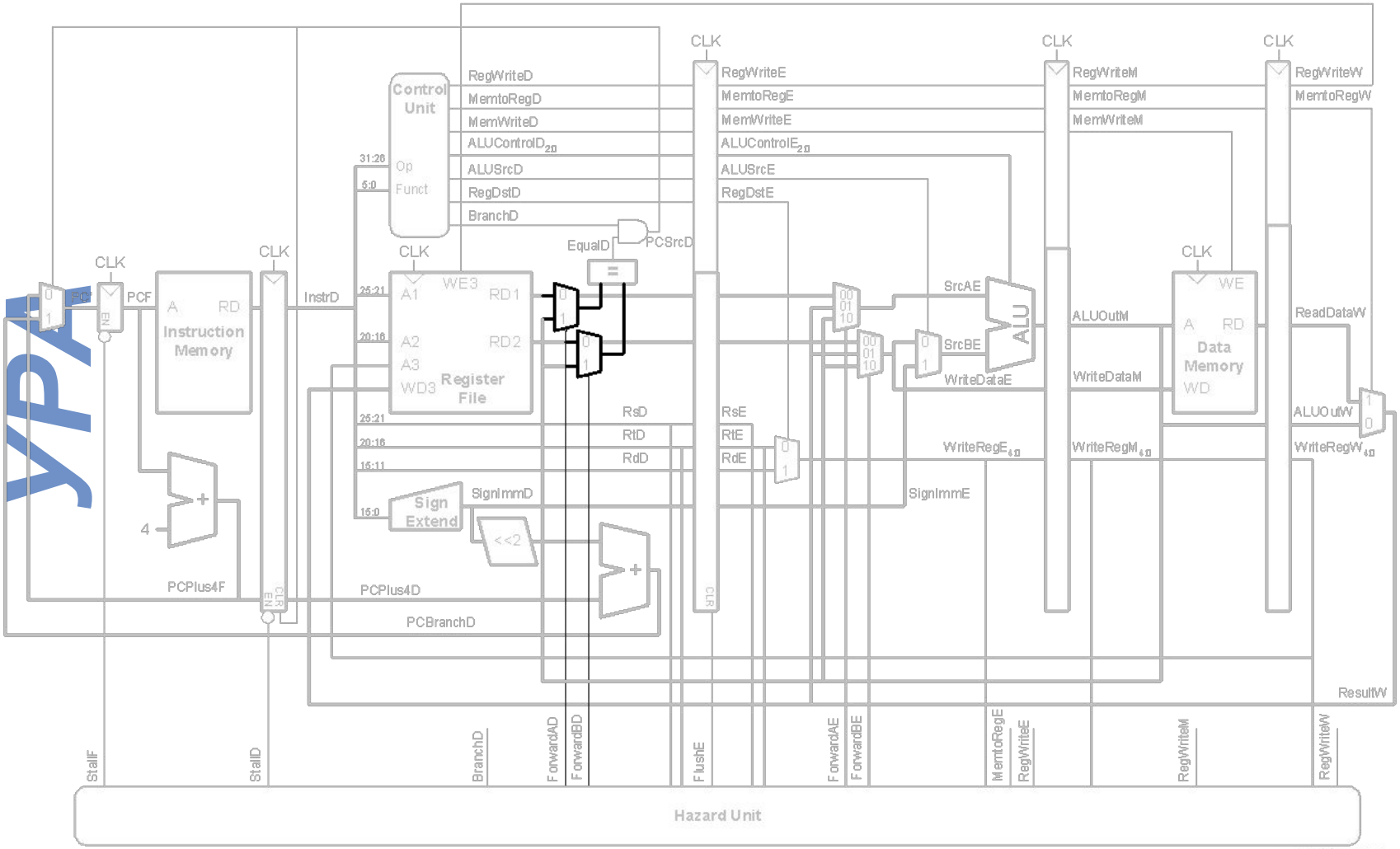
Приводит к новому конфликту данных на стадии Декодирования

Ранняя проверка условия



Инструкцию загруженную в конвейер после `beq` не обязательно удалять в случае выполнения перехода. Можно ввести условие, что инструкция следующая за переходом (условным или безусловным) выполняется всегда. Такое допущение называется branch delay slot.

Устранение конфликтов управления и данных



Логика устранения конфликтов

- **Логика управления передачей данных между стадиями конвейера (Forwarding logic):**

```
ForwardAD = (rsD !=0) AND (rsD == WriteRegM) AND RegWriteM
ForwardBD = (rtD !=0) AND (rtD == WriteRegM) AND RegWriteM
```

- **Логика останова конвейера (Stalling logic):**

```
branchstall = BranchD AND RegWriteE AND
                (WriteRegE == rsD OR WriteRegE == rtD)
              OR
        BranchD AND MemtoRegM AND
                (WriteRegM == rsD OR WriteRegM == rtD)

StallF = StallD = FlushE = lwstall OR branchstall
```

Предсказание переходов

- Мы можем попробовать оценить на сколько вероятно выполнение условного перехода и использовать наиболее вероятный результат
  - Например, в циклах наиболее вероятно выполнение условных переходов назад (переходы на начало итерации цикла скорее выполняются, чем нет)
  - Для улучшения предсказания переходов можно использовать результаты предыдущих предсказаний (например, если три прошлых раза мы переходили назад, то скорее всего это цикл и в следующий раз условный переход назад тоже состоится)
- Хорошее предсказание уменьшает количество сбросов стадий конвейера

Оценим производительность конвейерного процессора

- Тестовый набор SPECINT2000 содержит :
  - 25% инструкций lw
  - 10% инструкций sw
  - 11% условных переходов
  - 2% безусловных переходов
  - 52% инструкций R-типа
- Предположим:
  - 40% считываний из памяти используются следующей инструкцией
  - 25% переходов предсказываются неверно
  - Все переходы удаляют следующую инструкцию из конвейера
- **Каков средний CPI?**

Оценим производительность конвейерного процессора

- Тестовый набор SPECINT2000 содержит :
  - 25% инструкций lw
  - 10% инструкций sw
  - 11% условных переходов
  - 2% безусловных переходов
  - 52% инструкций R-типа
- Предположим:
  - 40% считываний из памяти используются следующей инструкцией
  - 25% переходов предсказываются неверно
  - Все переходы удаляют следующую инструкцию из конвейера
- **Каков средний CPI?**
  - Для `lw`/`beq` CPI = 1 если нет останова, 2 в случае останова
  - $CPI_{lw} = 1(0.6) + 2(0.4) = 1.4$
  - $CPI_{beq} = 1(0.75) + 2(0.25) = 1.25$

$$\textbf{Средний CPI} = (0.25)(1.4) + (0.1)(1) + (0.11)(1.25) + (0.02)(2) + (0.52)(1)$$

$$= \mathbf{1.15}$$

Оценим производительность конвейерного процессора

- Задержка самой длинной цепи комбинационной логики конвейерного процессора:

$$T_c = \max \{ \begin{array}{l} t_{pcq} + t_{\text{mem}} + t_{\text{setup}} \\ 2(t_{RFread} + t_{\text{mux}} + t_{\text{eq}} + t_{\text{AND}} + t_{\text{mux}} + t_{\text{setup}}) \\ t_{pcq} + t_{\text{mux}} + t_{\text{mux}} + t_{\text{ALU}} + t_{\text{setup}} \\ t_{pcq} + t_{\text{memwrite}} + t_{\text{setup}} \\ 2(t_{pcq} + t_{\text{mux}} + t_{\text{RFwrite}}) \} \end{array}$$

Оценим производительность конвейерного процессора

Параметр	Обозначение	Задержка (пс)
Время записи в регистр	t_{pcq_PC}	30
Время предустановки регистра	t_{setup}	20
Задержка мультиплексора	t_{mux}	25
Задержка АЛУ	t_{ALU}	200
Задержка считывания из памяти	t_{mem}	250
Задержка считывания из рег. файла	t_{RFread}	150
Время предустановки рег. файла	$t_{RFsetup}$	20
Задержка компаратора	t_{eq}	40
Задержка элемента И	t_{AND}	15
Задержка записи в память	$T_{memwrite}$	220
Задержка записи в рег. файл	$t_{RFwrite}$	100 ps

$$T_c = 2(t_{RFread} + t_{mux} + t_{eq} + t_{AND} + t_{mux} + t_{setup})$$
$$= 2[150 + 25 + 40 + 15 + 25 + 20] \text{ ps} = \mathbf{550\ пс}$$

Оценим производительность конвейерного процессора

Предположим, в программе 100 миллиардов инструкций

Время выполнения = (# инструкции) × CPI × T_c

$= (100 \times 10^9)(1.15)(550 \times 10^{-12})$

= 63 секунды

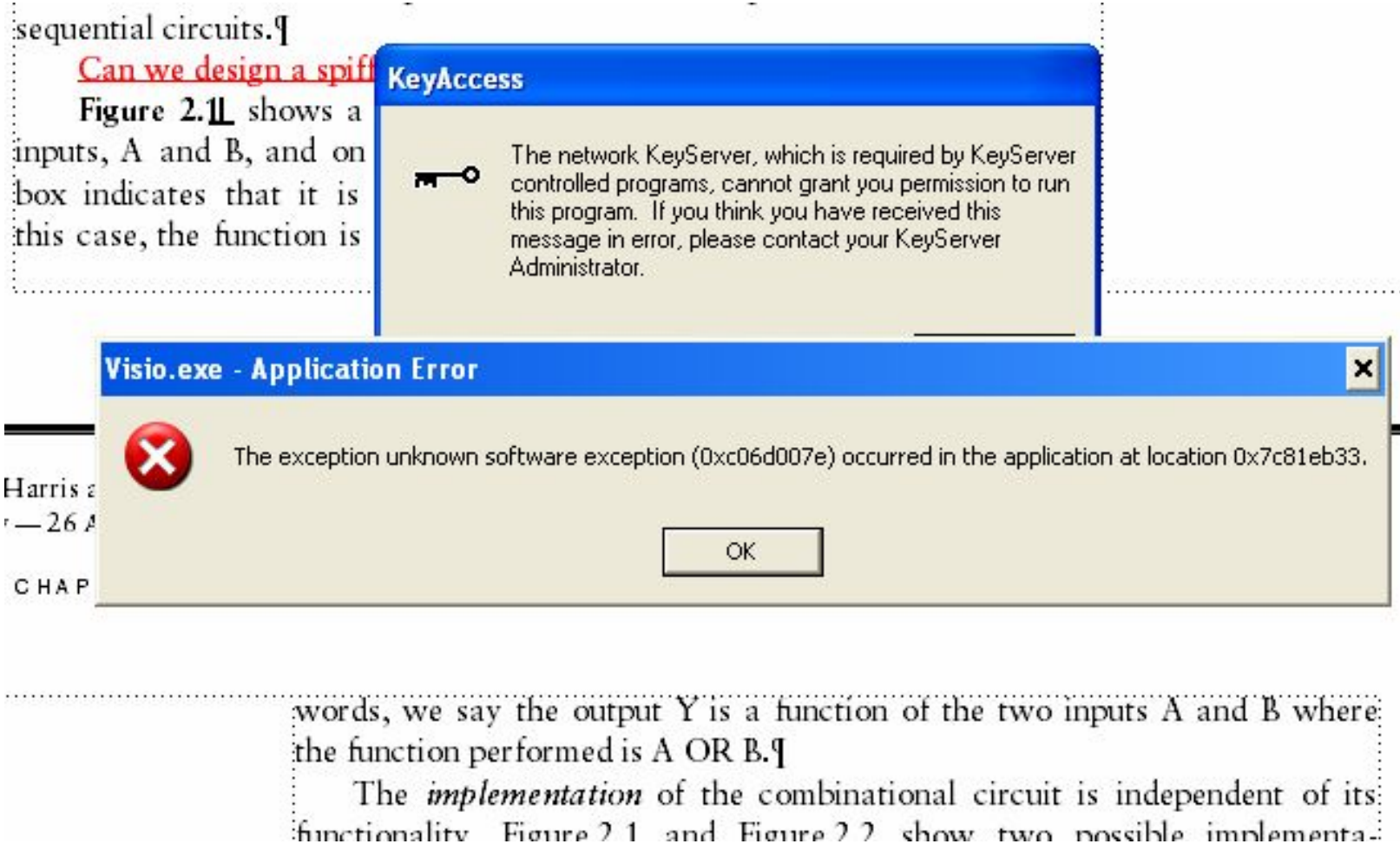
Сравнение

Микроархитектура	Время выполнения (секунды)	Прирост производительности
Однотактная	92.5	1
Многотактная	133	0.70
Конвейерная	63	1.47

Повторение: Исключения

- Исключение (англ.: exception) – незапланированный вызов функции-обработчика исключения (*exception handler*)
- Исключения бывают:
  - Аппаратные (например, нажатие клавиши на клавиатуре). Такие исключения называют *прерываниями (*англ.: *interrupt)*.
  - Программные (например, неизвестная инструкция, деление на ноль). Такие исключения называют *ловушками (*англ.: *trap).*
- При возникновении исключения процессор:
  - Записывает в специальный регистр код причины исключения
  - Сохраняет значение счетчика команд (PC) на момент возникновения исключения (адрес инструкции вызвавшей искл.)
  - Делает переход в функцию-обработчик исключения по адресу 0x80000180
  - После выполнения обработчика исключения возвращает управление прерванной программе, делая переход по ранее сохраненному адресу

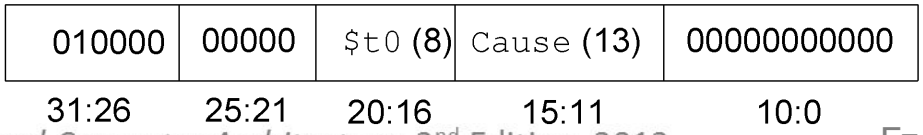
Пример исключения



Регистры обработки

- Отдельные регистры. Не входят в регистровый файл
  - `Cause`
    - Содержит код причины исключения
    - Регистр 13 Сопроцессора 0
  - `EPC` (Exception PC)
    - Содержит значение счетчика команд (PC) на момент возникновения исключения
    - Регистр 14 Сопроцессора 0
- Инструкция считывания регистра Сопроцессора 0 в регистр общего назначения
  - `mfc0 $t0, Cause`
  - Копирует содержимое `Cause` в регистр `$t0`

mfc0

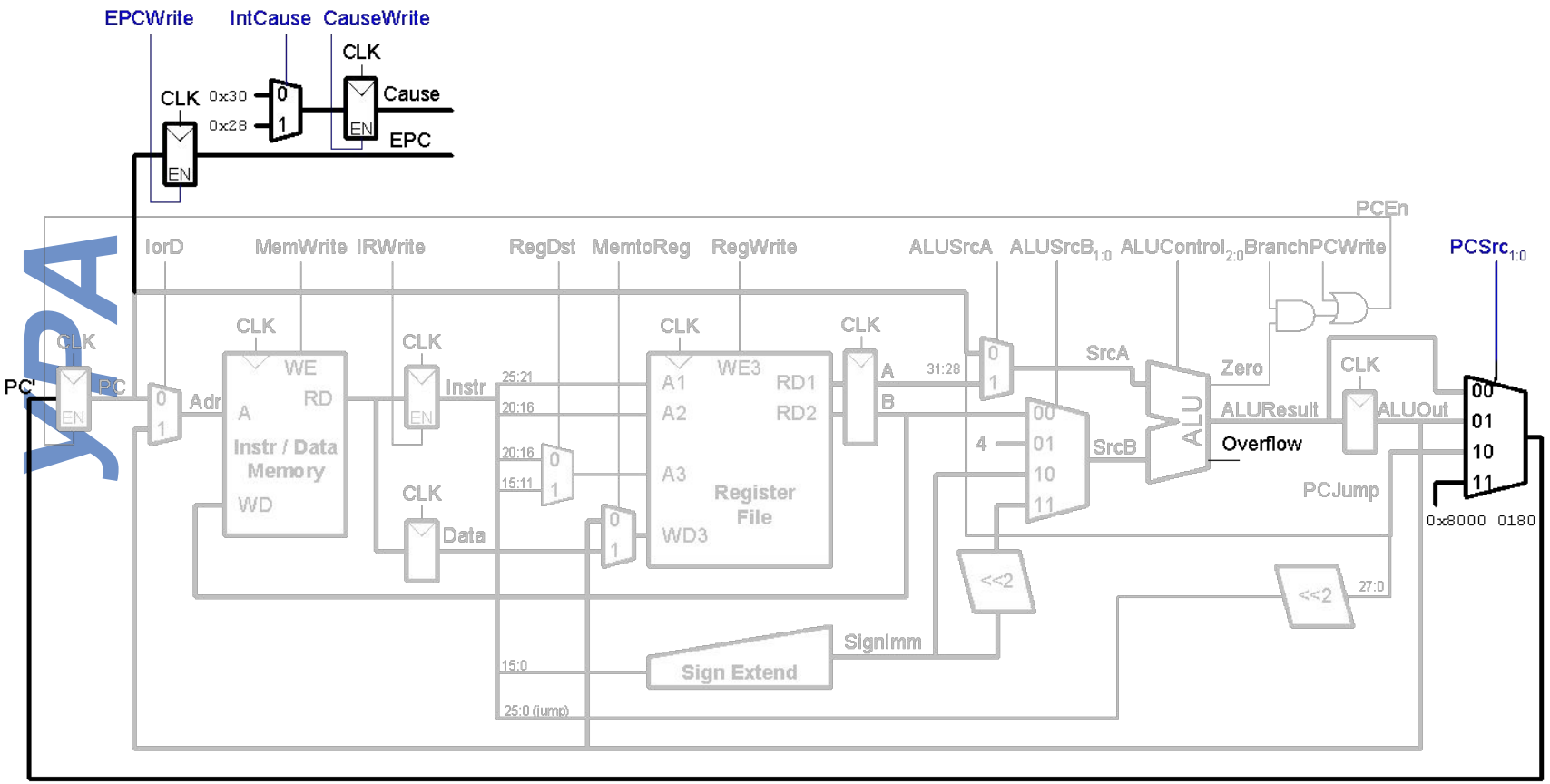


Коды причин исключений

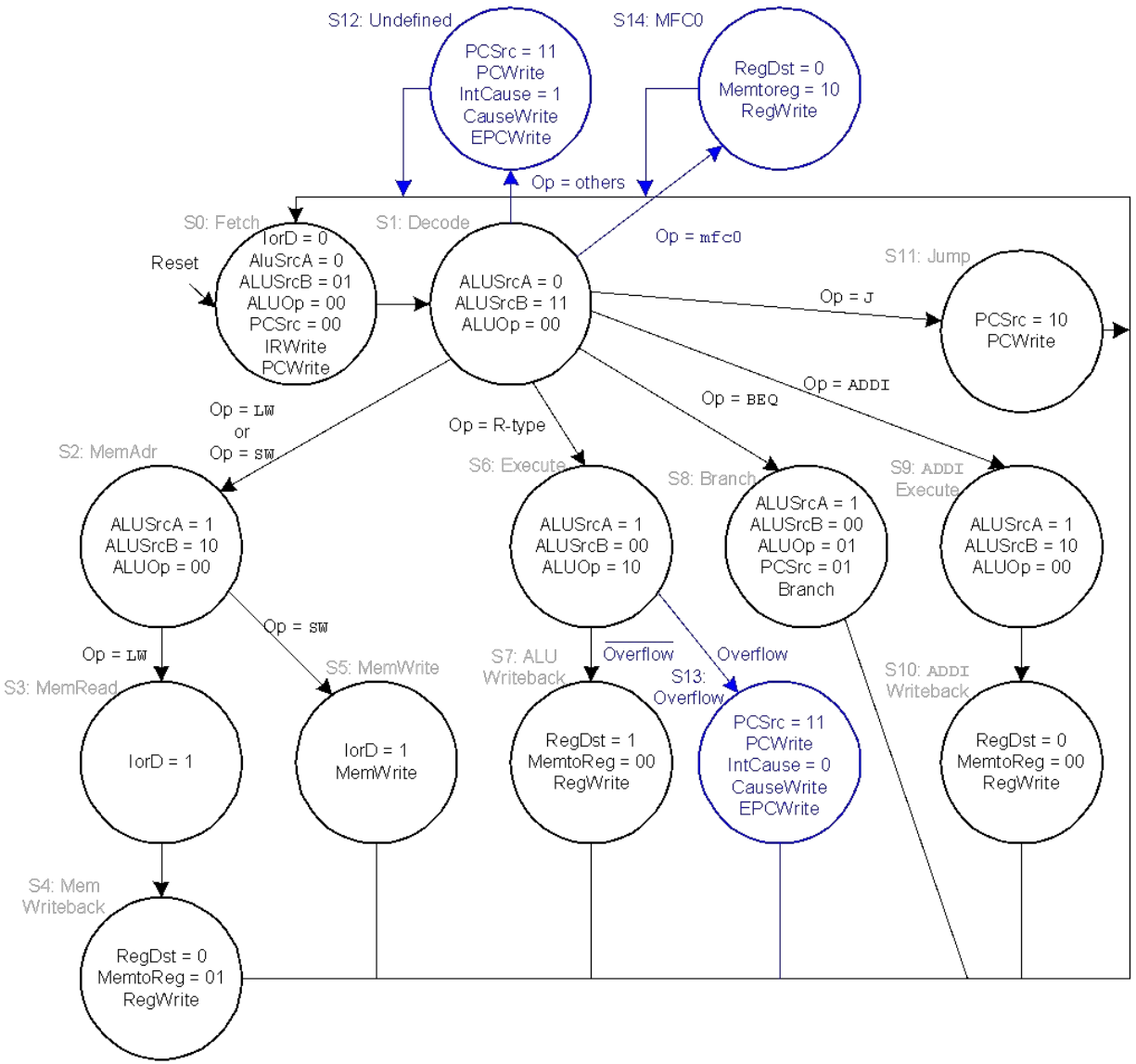
Исключение	Код причины
Аппаратное прерывание	0x00000000
Системный вызов	0x00000020
Точка останова / Деление на 0	0x00000024
Неизвестная инструкция	0x00000028
Арифметическое переполнение	0x00000030

Добавим в многотактный MIPS процессор возможность обработки последних двух типов исключений

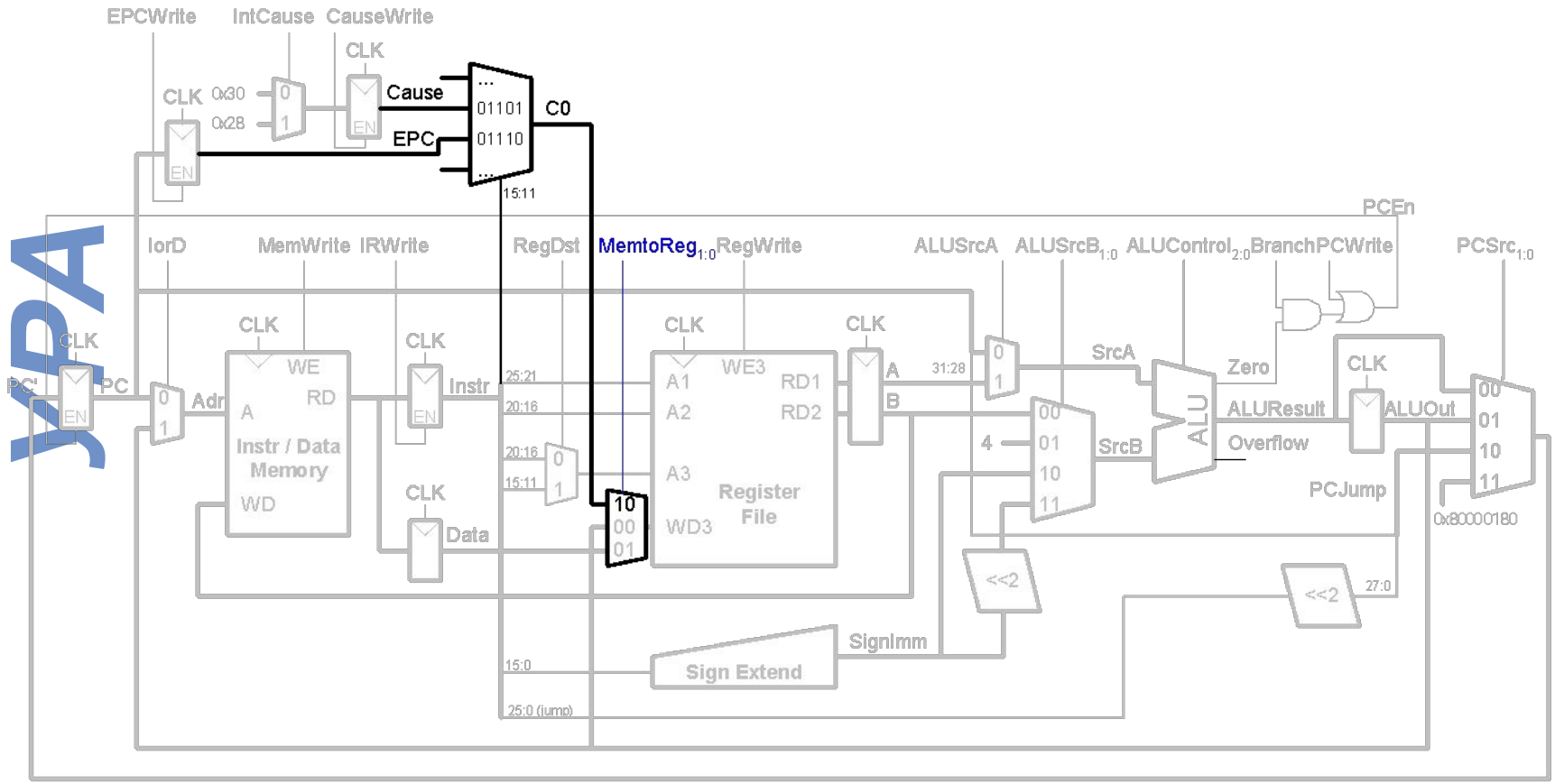
Аппаратура исключений: EPC &



Исключения в управляющем автомате



Аппаратура исключений:



Улучшения

- Длинные конвейеры
- Динамическое предсказание переходов
- Суперскалярные процессоры
- Процессоры с внеочередным выполнением инструкций
- Переименование регистров
- SIMD
- Многопоточность
- Многопроцессорность

Длинные конвейеры

- Содержат 10-20 стадий
- Количество стадий ограничивается:
  - Конфликтами конвейера
  - Энергопотреблением
  - Стоимостью
  - Увеличением задержки тактового сигнала

Предсказание переходов

- У идеального конвейерного процессора: CPI = 1
- Неверное предсказание переходов увеличивает CPI
- **Статическое предсказание переходов:**
  - Проверяем направление перехода (вперед или назад)
  - Если переход назад, считаем, что он будет выполнен
  - Иначе, считаем, что переход не будет выполнен
- **Динамическое предсказание переходов:**
  - Процессор содержит таблицу с последними несколькими сотнями (или тысячами) инструкций условного перехода. Эту таблица иногда называют буфером целевых адресов ветвлений (branch target buffer). Она содержит адреса переходов и информацию о том, был ли переход выполнен

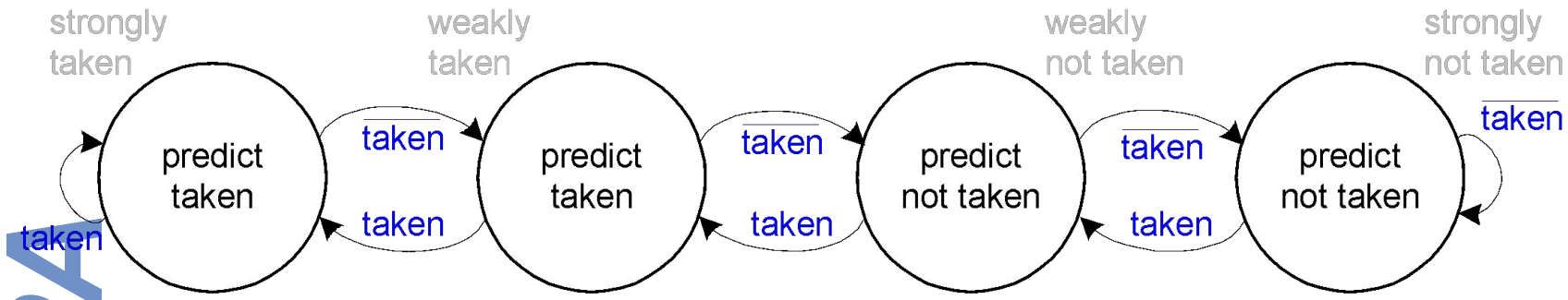
Пример предсказания

```
    add  $s1, $0, $0       # sum = 0
    add  $s0, $0, $0       # i   = 0
    addi $t0, $0, 10       # $t0 = 10
for:
    beq  $s0, $t0, done    # if i == 10, branch
    add  $s1, $s1, $s0     # sum = sum + i
    addi $s0, $s0, 1       # increment i
    j    for
done:
```

Однобитный динамический предсказатель переходов

- Запоминает, был ли переход выполнен в прошлый раз, и предсказывает, что в следующий раз произойдет то же самое
- Ошибается дважды: для первого и последнего условных переходов цикла (на первой и последней итерации)

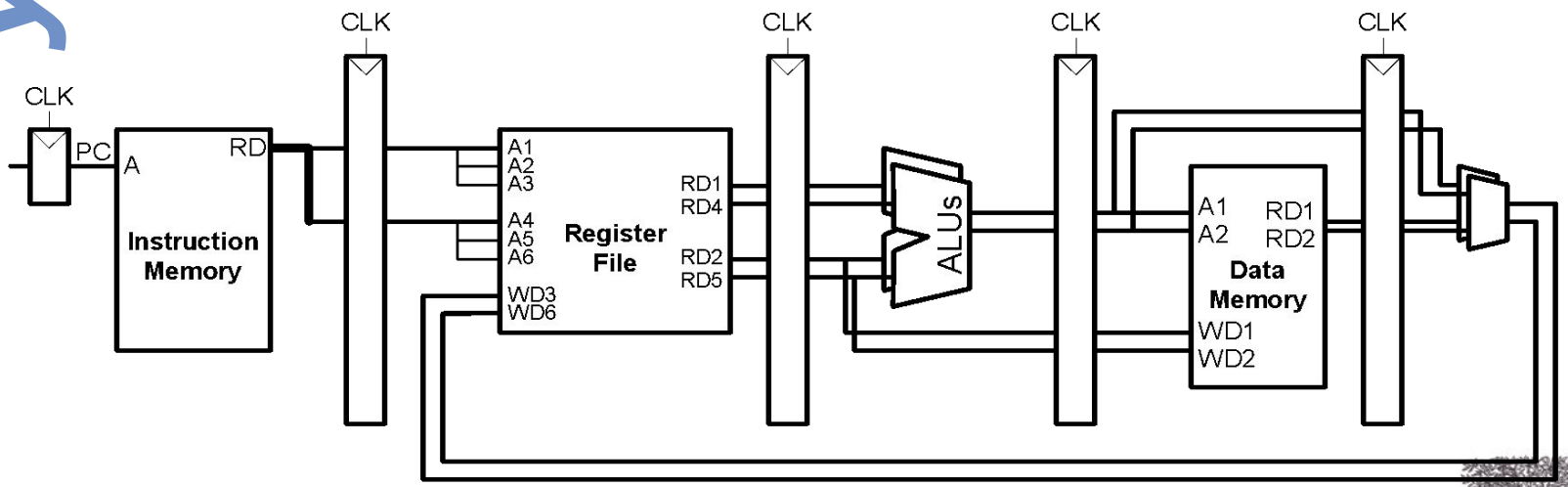
Двухбитный динамический предсказатель переходов



Дает неверное предсказание только для последнего условного перехода цикла (на последней итерации)

Суперскалярный процессор

- Позволяет одновременно считывать и выполнять несколько инструкций за счет дублирования функциональных блоков
- Как будто в процессоре одновременно функционирует несколько конвейеров
- Зависимости между инструкциями значительно усложняют их одновременное выполнение

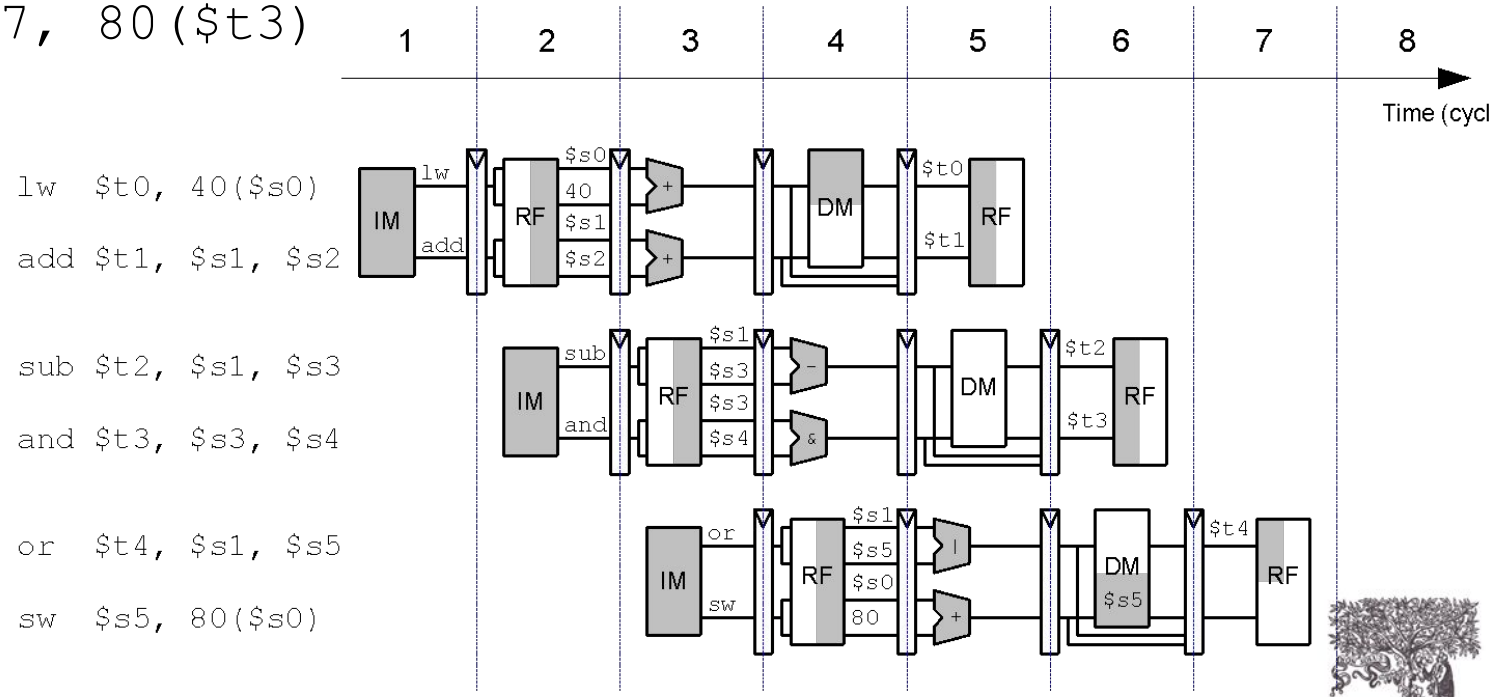


Пример суперскалярности

```
lw  $t0, 40($s0)
add $t1, $t0, $s1
sub $t0, $s2, $s3
and $t2, $s4, $t0
or  $t3, $s5, $s6
sw  $s7, 80($t3)
```

Идеальный IPC: 2

Реальный IPC: 2

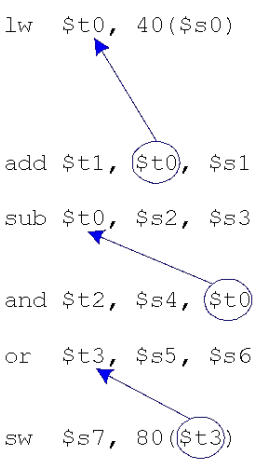


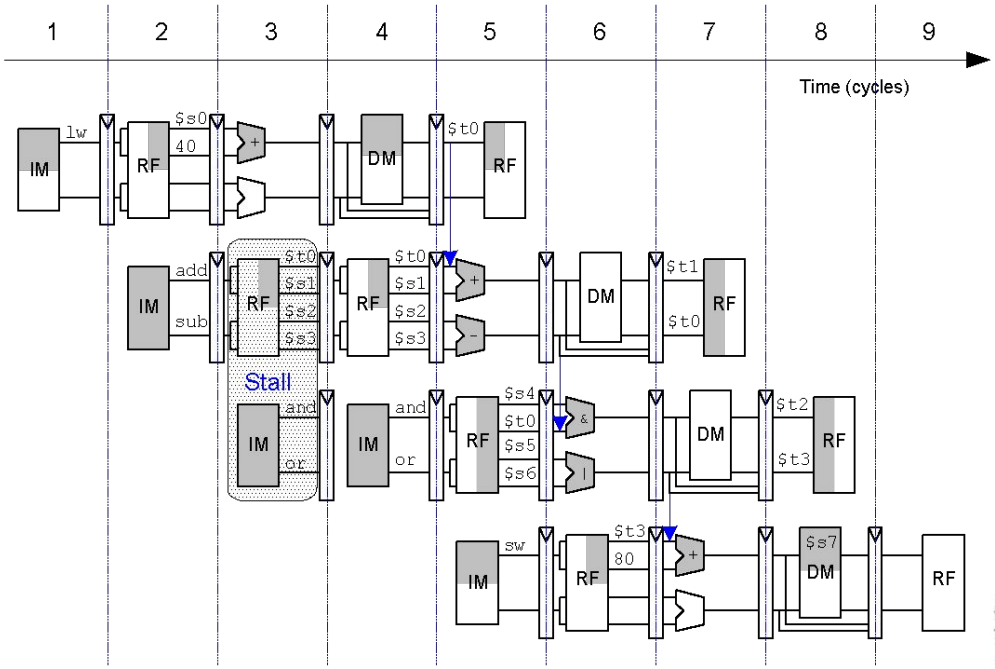
Суперскалярность с

```
lw   $t0, 40($s0)
add  $t1, $t0, $s1
sub  $t0, $s2, $s3
and  $t2, $s4, $t0
or   $t3, $s5, $s6
sw   $s7, 80($t3)
```

Идеальный IPC: 2

Реальный IPC: 6/5 = 1.17





Внеочередное выполнение инструкций

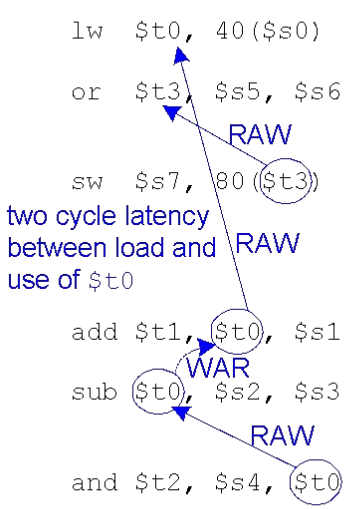
- Процессор заранее просматривает наперед большое количество инструкций, находит независимые друг от друга инструкции и запускает их на одновременное выполнение
- Инструкции могут выполняться не в том порядке, в котором они расположены в программе
- Процессор следит за тем, чтобы внеочередное выполнение не нарушало алгоритм работы программы
- **Зависимости:**
  - **RAW** (read after write, чтение после записи): предыдущая инструкция записывает, следующая считывает регистр
  - **WAR** (write after read, запись после чтения): предыдущая инструкция считывает регистр, следующая инструкция записывает этот регистр
  - **WAW** (write after write, запись после записи): инструкция пытается писать в регистр после того, как в него уже записала следующая по ходу программы инструкция

Внеочередное выполнение инструкций

- **Параллелизм на уровне инструкций (Instruction level parallelism, ILP):** число инструкций, которые могут выполнятся одновременно (обычно < 3)
- **Таблица готовности (Scoreboard):** таблица, хранящая информацию про:
  - Инструкции ожидающие выполнения
  - Доступные функциональные блоки (АЛУ, порты памяти и т.д.)
  - Зависимости между инструкциями

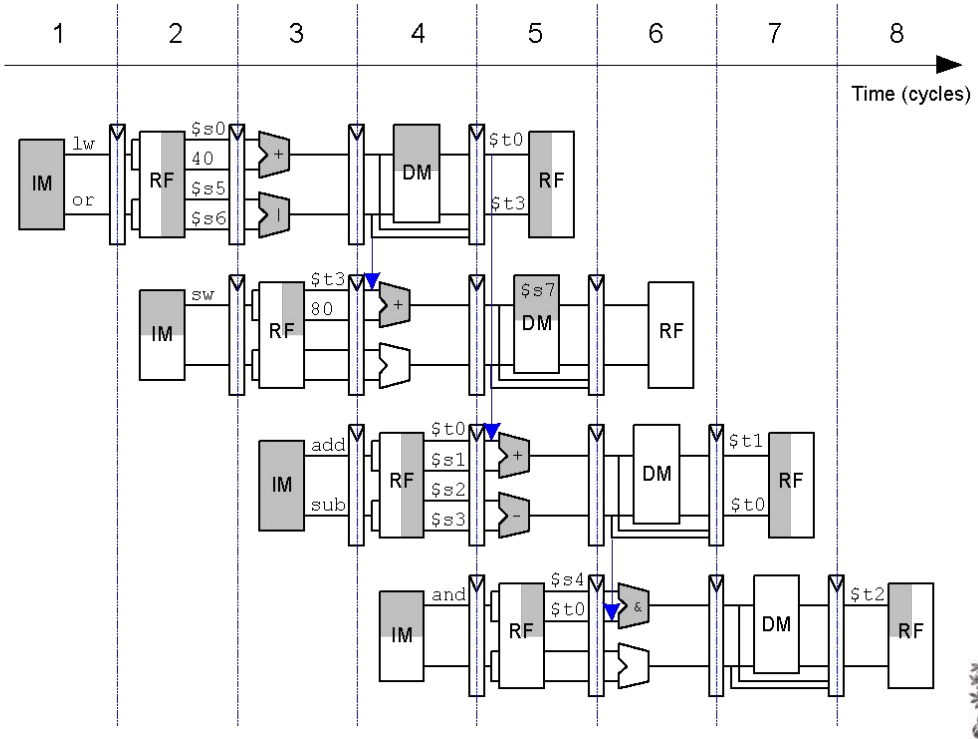
Пример внеочередного выполнения

```
lw  $t0, 40($s0)
add $t1, $t0, $s1
sub $t0, $s2, $s3
and $t2, $s4, $t0
or  $t3, $s5, $s6
sw  $s7, 80($t3)
```



Идеальный IPC: 2

Реальный IPC: 6/4 = 1.5

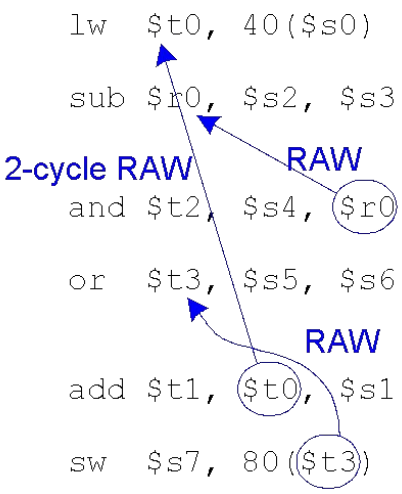


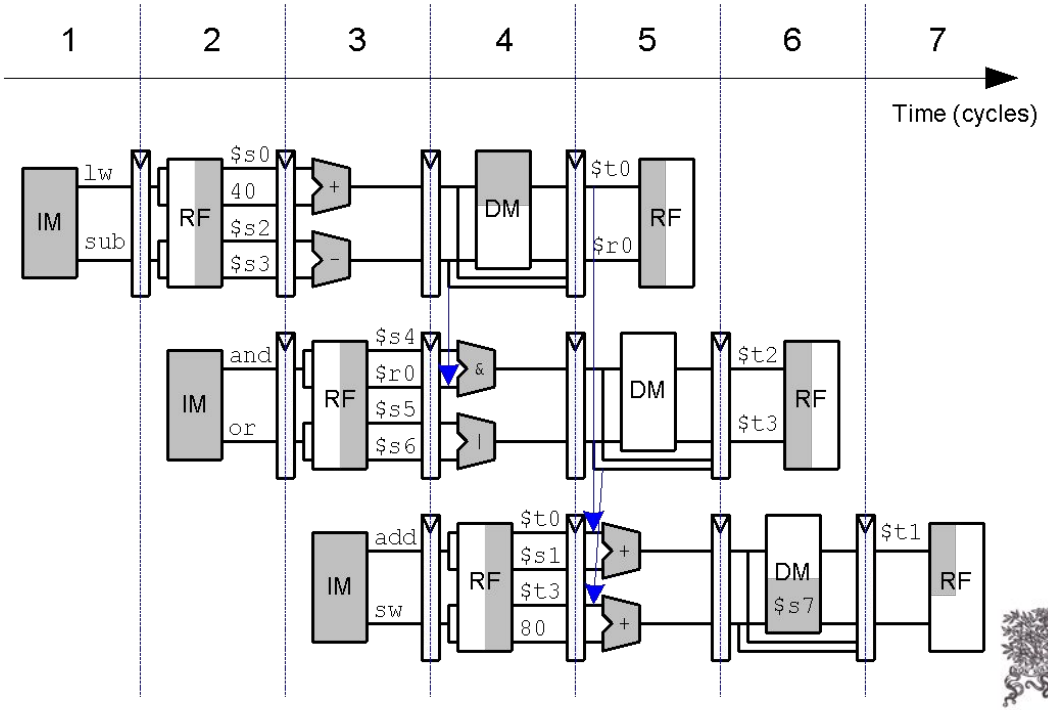
Переименование регистров

```
lw  $t0, 40($s0)
add $t1, $t0, $s1
sub $t0, $s2, $s3
and $t2, $s4, $t0
or  $t3, $s5, $s6
sw  $s7, 80($t3)
```

Идеальный IPC: 2

Реальный IPC: 6/3 = 2





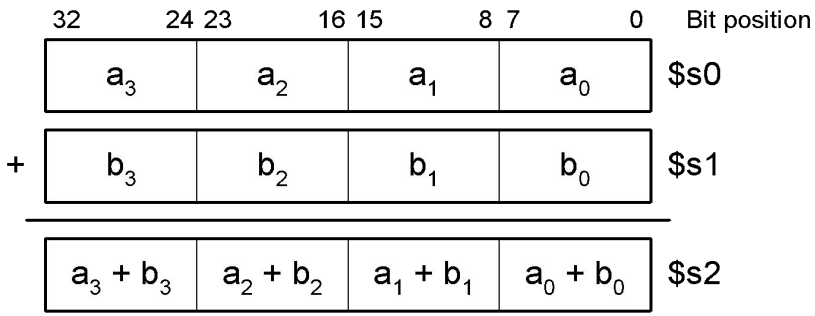
МИКРОАРХИТЕКТУРА

ELSEVIER

SIMD

- Одиночный поток команд, множественный поток данных (Single Instruction Multiple Data, SIMD)
  - Одна инструкция обрабатывает множество блоков данных одновременно (например, параллельно суммирует несколько пар чисел)
  - Часто используется в компьютерной графике
  - Выполняется арифметическая операция над несколькими небольшими независимыми блоками данных (пакованная арифметика)
- Например, в 32-разрядном сумматоре можно одновременно суммировать 4-ре пары 8-битных операндов

```
padd8 $s2, $s0, $s1
```



Многопоточность и мультипроцессорность

- **Многопоточность**
  - Например, в текстовом редакторе один поток может отвечать за обработку вводимых с клавиатуры символов и набор текста, другой поток “одновременно” выполнять проверку правописания, третий поток может при этом выводить текст на печать
- **Мультипроцессорность**
  - Несколько отдельных процессоров внутри одного чипа

Потоки: Определения

- **Процесс:** программа, которая выполняется на компьютере
  - Несколько процессов могут выполняться одновременно, например: веб серфинг, прослушивание музыки, написание статьи в текстовом редакторе
- **Поток:** часть процесса (программы)
  - Процесс может содержать несколько потоков, например текстовый редактор может содержать потоки для набора текста, проверки орфографии, печати

Потоки в обычном процессоре

МИКРОАРХИТЕКТУРА

- В каждый момент времени выполняется один поток
- Когда выполнение потока блокируется (например, поток ожидает данные из медленной внешней памяти):
  - Архитектурное состояние потока сохраняется
  - Архитектурное состояние следующего потока загружается в процессор и поток запускается на выполнение
  - Такая процедура называется **переключением контекста**
- До тех пор, пока процессор переключается между потоками достаточно быстро, пользователю кажется, что все потоки выполняются одновременно.

Многопоточность

- У многопоточного процессора есть несколько копий архитектурного состояния
- Несколько потоков могут быть **активны** одновременно:
  - Когда выполнение одного потока блокируется, сразу же запускается выполнение другого потока на имеющихся функциональных блоках
  - Если один поток не использует все функциональные блоки процессора, их использует другой поток
- Многопоточность не влияет на параллелизм на уровне инструкций (ILP) отдельного потока, но увеличивает общую производительность вычислений

Intel называет такую технологию “hyperthreading”

Многопроцессорность

- Многопроцессорная система (multiprocessor system), или просто мультипроцессор, состоит из нескольких процессоров и аппаратуры для соединения их между собой
- Типы:
  - **Гомогенная (симметричная) многопроцессорность**: несколько одинаковых процессоров подключены к общей памяти
  - **Гетерогенная (асимметричная) многопроцессорность:** разные типы процессорных ядер используются для задач разных типов (например, в мобильном телефоне для вычислений используется обычный процессор, а для обработки аудио/видео – специализированное DSP ядро)
  - **Кластеры:** каждое ядро имеет свою собственную память

Дополнительные ресурсы

- Patterson & Hennessy's: *Computer Architecture: A Quantitative Approach*
- Conferences:
  - www.cs.wisc.edu/~arch/www/
  - ISCA (International Symposium on Computer Architecture)
  - HPCA (International Symposium on High Performance Computer Architecture)