Лекция 8. Схемы построения однотактного процессора.

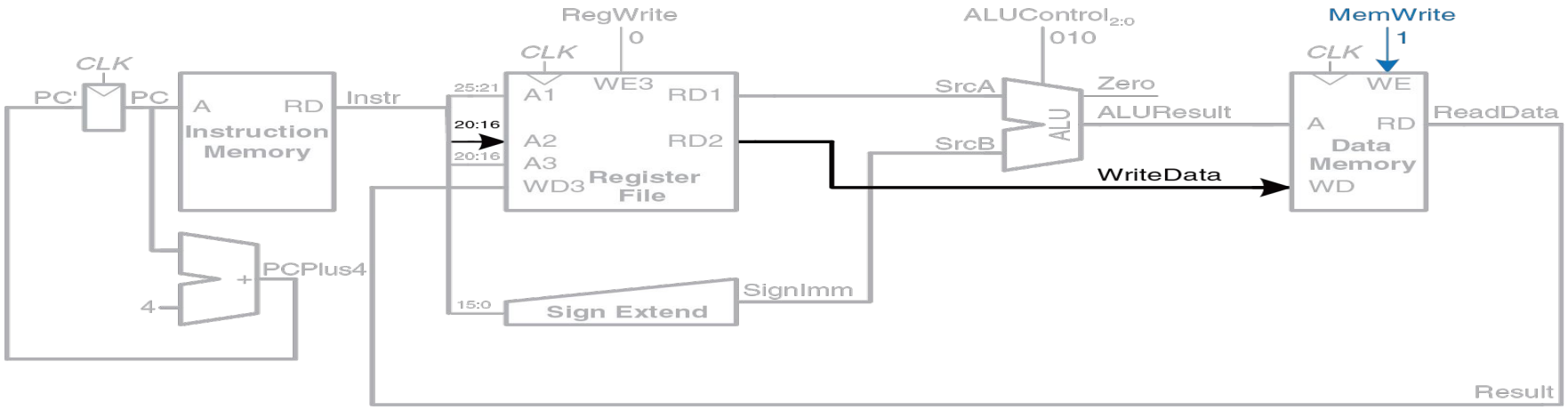
Вопросы:

1. Особенности работы схемы при выполнении команд **Lw** и **sw**.
2. Микроархитектура с тремя мультиплексорами для обработки команд типа **R**.
3. Микроархитектура с командами условного перехода.
4. Однотактное устройство управления.
5. Анализ производительности однотактного процессора.

Литература: Дэвид М. Харрис и Сара Л. Харрис. Цифровая схемотехника и архитектура компьютера, второе издание, с. 938 -966.

1. Особенности схемы построения микроархитектуры процессора при выполнении команд **Lw** и **sw**.

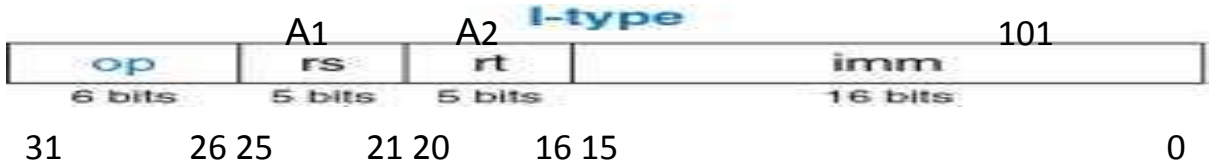
1.1. Тракт записи данных команды **SW** в память.



Команда

sw

Sw $s3,4($0)

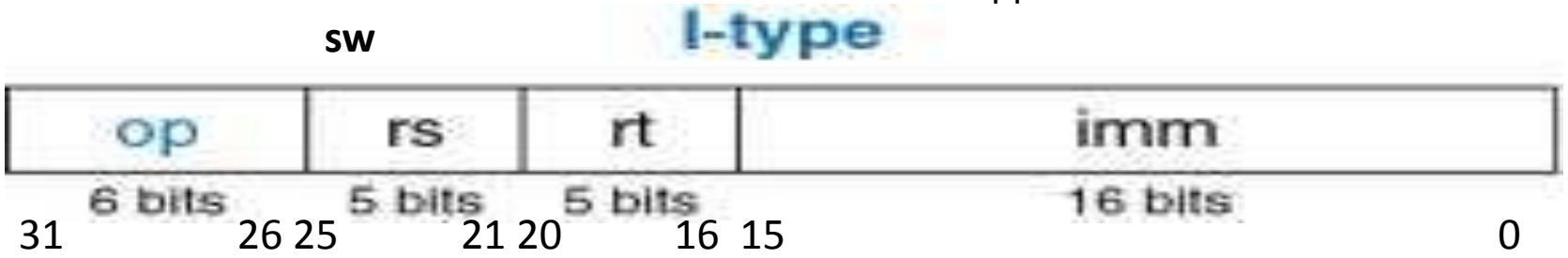


Команда **sw** для определения адреса записи **A** в **DM** читает из файла регистров по адресу **A1** составляющую адреса **ScrA** из **RD1** и число **4** из поля **imm** для составляющей адреса **ScrB.** **Для определения слова, которое надо записать** в **DM** читается еще один регистр по адресу **A2** из порта **RD2** **RF** и его содержимое записывает в память данных.

Адрес регистра **A2** указывается в поле **rt** (**Instr20:16**). Эти пять бит подключены по **A2** ко второму порту регистрового файла **RD2**. Прочитанное значение из **RF** появляется на выходе **RD2** и попадает на вход записи **WD** в память данных **DM**. Вход разрешения записи **WE** управляется сигналом MemWrite. Для команды **sw** сигнал MemWrite = 1, чтобы данные были записаны в память **DM**.

Для команды **Lw** всегда сигнал MemWrite = 0 и данные не записываются в память **DM**.

Особенности исполнения команд типа **Lw** и **sw**



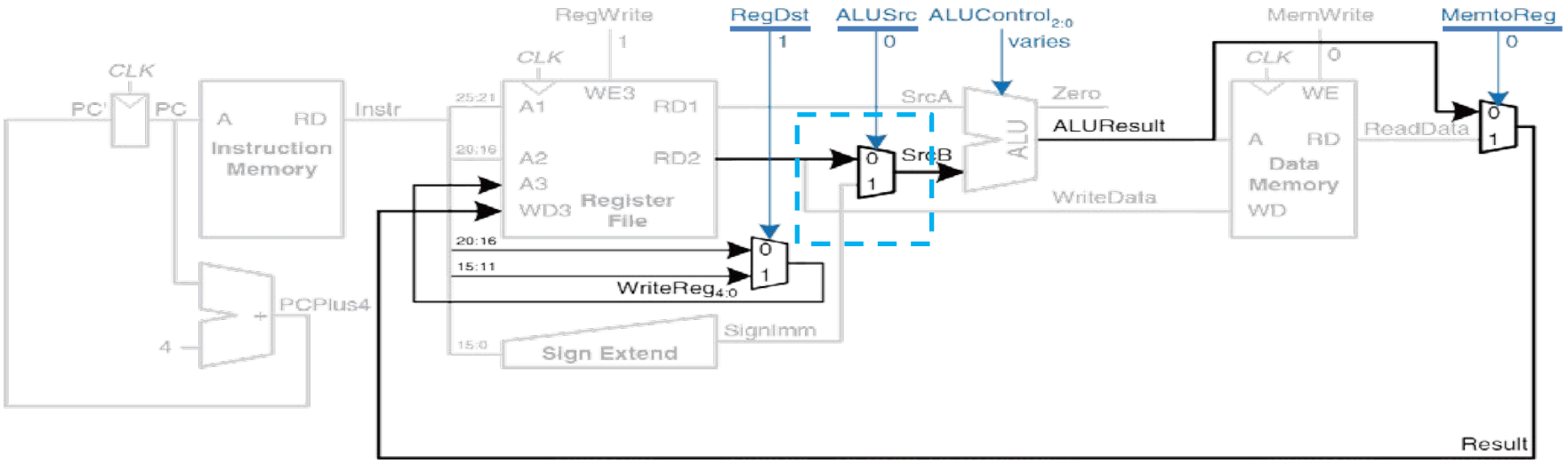
1. Как и команда **загрузка слова** (**Lw**), так и команда **сохранения слова** (**sw**), работают с **расширением** и читают **базовый адрес** из поля команды **rs** (Instr 25:21) по адресу **A1** с первого порта **RD1** файла регистров **RF.** Этот адрес определяет непосредственный операнд, читаемый из регистра **RD1 регистрового файла RF .** Кроме того, данная команда определяет знаковое смещение **Sign Extend** (Instr 15:0), которое находится в младших 16 битах команд. АЛУ складывает *базовый адрес* с *расширенным смещением* и получает **адрес A** для памяти данных **DM, где надо** *прочитать* для команды **Lw** слово или **куда надо** *записать* для команды **sw** слово. **Эта операция одинаково реализуется в тракте данных как для команды Lw , так и для команды sw.**

2. Команда **sw**, **ничего не записывает в файл регистров** в отличие от **Lw**. Команда **sw** читает из инструкции еще одно поле **rt** (Instr 20:16) и определяет его содержимое по адресу **A2 регистрового файла.** Эти пять бит подключены ко второму порту **A2 RF,** которые определяют адрес чтения и на выходе **RD2** файла регистров **RF** появляется **32 – битный операнд.** Он попадает на вход записи **WD** в **память данных DM**. Вход разрешения записи (**WE**) управляется сигналом MemWrite записи памяти. Для команды **sw** сигнал MemWrite = 1, чтобы данные были записаны в память данных **DM**. При этом **управляющие сигналы**: ALUControl = 010, чтобы базовый адрес был просуммирован со смещением, *а значение* ***Read Data*** из памяти данных **DM** ***в этом случае*** *игнорируется, так как в регистровом файле* ***RF*** ***Reg Write****= 0.*

2. Микроархитектура с тремя мультиплексорами для обработки команд типа R.

Схема с тремя мультиплексорами **ALUSrc, MemotReg, RegDst** .

Тракт данных с поддержкой команд типа R .



Формат команды

6b 5b 5b 5b 5b 6b

31 26 25 21 20 16 15 11 10 6 5 0

1.Мультиплексор формирования составляющих адреса ALUSrc.

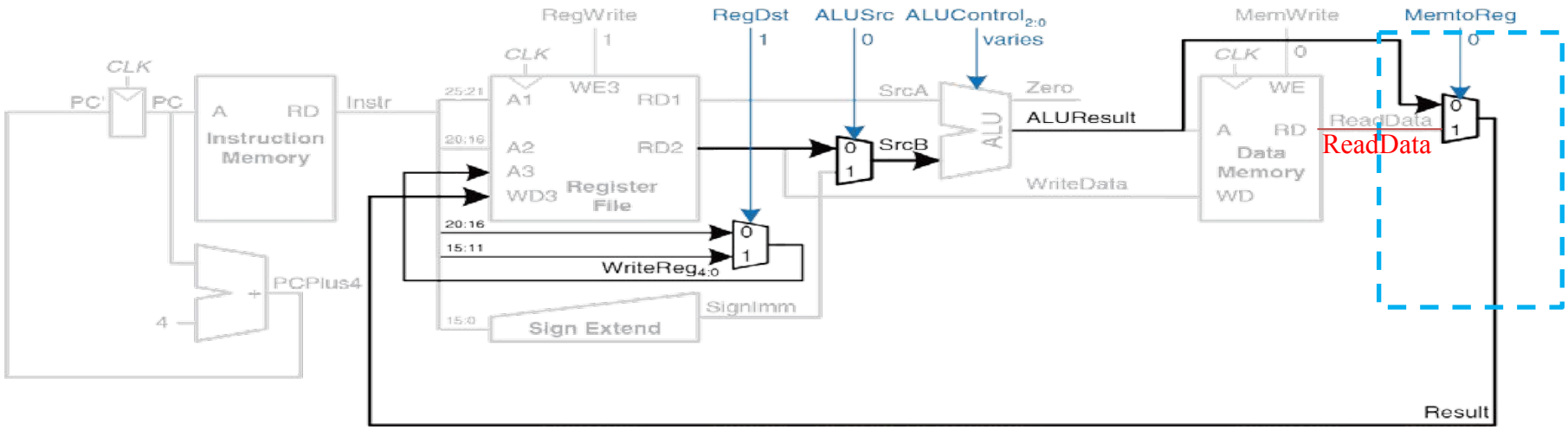
Содержимое двух регистров **RD1** и **RD2** читается из файла регистров по адресам **A1, A2** и подается на входы **АЛУ.** Ранее операнд **SrcB** вычислялся через непосредственное значению **SignImm** с расширением знака. Теперь добавляем мультиплексор, чтобы была возможность подать на вход **АЛУ** или **SignImm**, или выход **RD2** файла регистров **RF**.

Мультиплексор управляется новым сигналом **ALUSrc**. **ALUSrc** равен **нулю** для команд **типа R** и в этом случае на вход **АЛУ** подается значение из файла регистров.

2.Мультиплексор записи MemotReg.

При выполнении команды **Lw** порт записи **WD3** файла регистров **RF** подключался к памяти данных **DM** по выходу **RD** (сигнал ReadData).

Однако команды типа **R** так же должны записываться в **файл регистров RF** значение **ALUResult** по адресу**A3** поле команды **rd**. Чтобы выбирать между ReadData при выполнении команды **Lw** и **ALUResult** для команд типа **R**, добавим еще один мультиплексор **MemotReg**, выход которого соединен со входом записи **WD3 RF.**



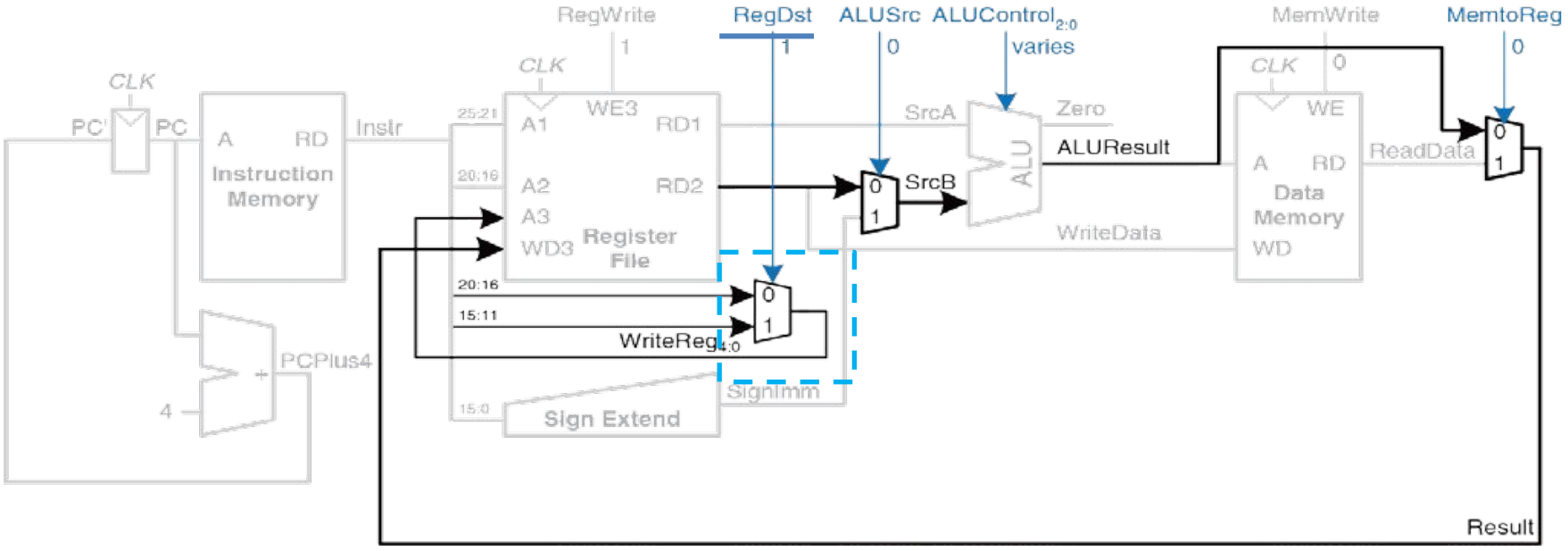
Если выход **MemtoReg =0** равен **нулю** для команд типа **R,** то в этом случае **Result** принимает

значение **ALUResult**. Для команды **Lw** **MemtoReg=1** равен **единице**, а **Result** принимает

значение **ReadData**. Для команды **sw** значение **MemotReg**, **равное 1** не играет никакой роли, так как **sw** ничего в регистровый файл не пишет.

3.Мультиплексор определения адреса записи RegDst .

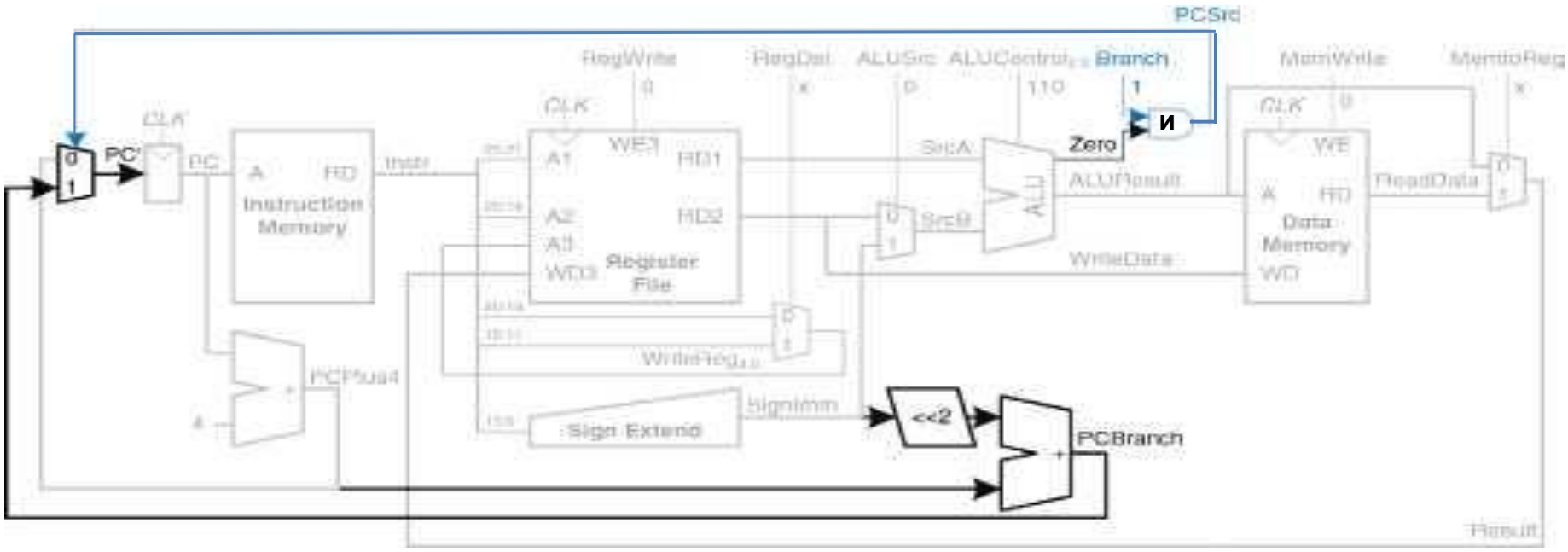
Адрес регистра в **файле регистров**, куда нужно было записать данные, определялся полем **rt (Instr 20:16)** для команды **Lw**. Однако, для команд типа **R**, адрес регистра задается полем **rd (Insr 15:11),** поэтому нужен **третий мультиплексор**, чтобы присваивать сигналу **WriteReg** значение из нужного поля. Этот мультиплексор управляется сигналом **RegDst**. **RegDst=1** для команд типа **R** - в этом случае **WriteReg** принимает значение поля **rd (Insftr 15:11).** Для команды **Lw RegDst=0**, а **WriteReg** принимает значение поля **rt (Instr 20:16).**



Примечание:

Для команды **sw** значение **RegDst** не играет никакой роли, так как **sw** ничего в регистровый файл не пишет

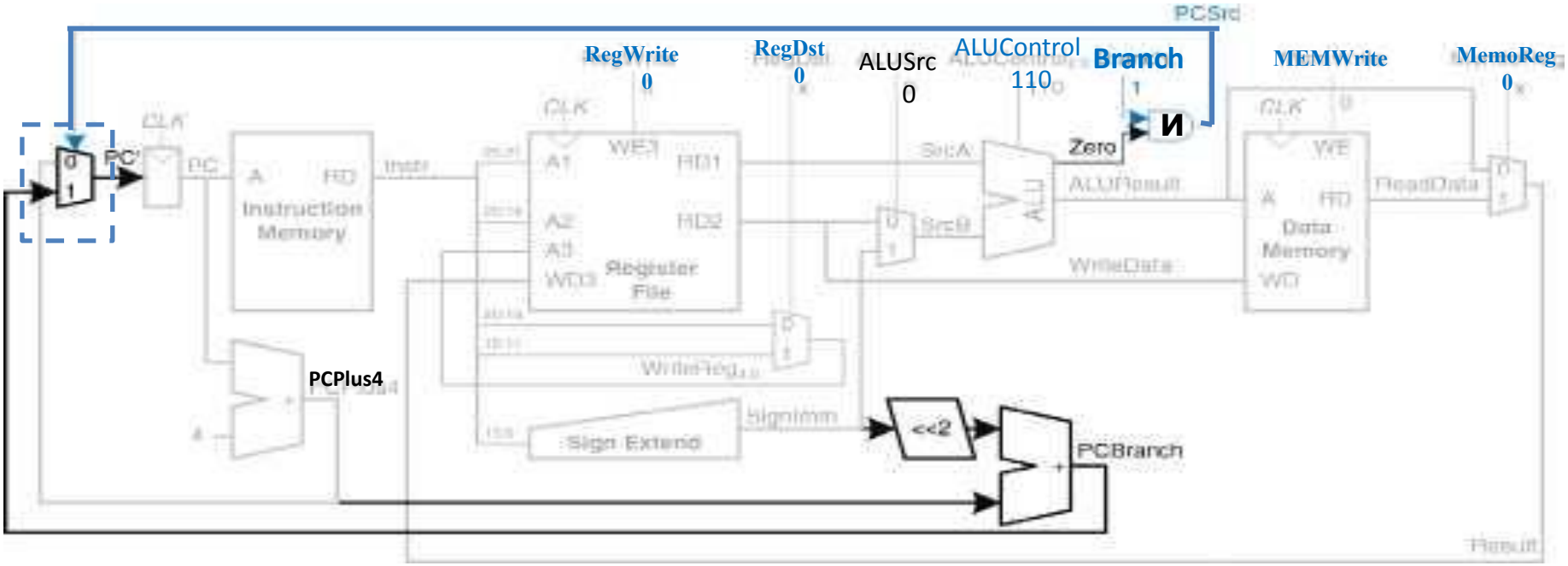
3. Микроархитектура с командами условного перехода.



Команда условного перехода beq- переход, если равно.

Эта команда ***сравнивает два регистра*** и, ***если они равны, то добавляет смещение к счетчику команд PC, выполняя, таким образом, условный переход***.

1. **Смещение** - это положительное или отрицательное число, передаваемое как непосредственный операнд в поле инструкции **imm: Instr15:0**. Смещение указывает количество команд, которое нужно пропустить, прежде чем продолжить выполнение программы. Над значением непосредственного операнда надо выполнить *операцию знакового расширения*, после чего умножить его (количество команд) на четыре, чтобы получить новое значение счетчика команд (следующий адрес после команды перехода плюс смещение): **PC' = PC + 4 + SignImm * 4**

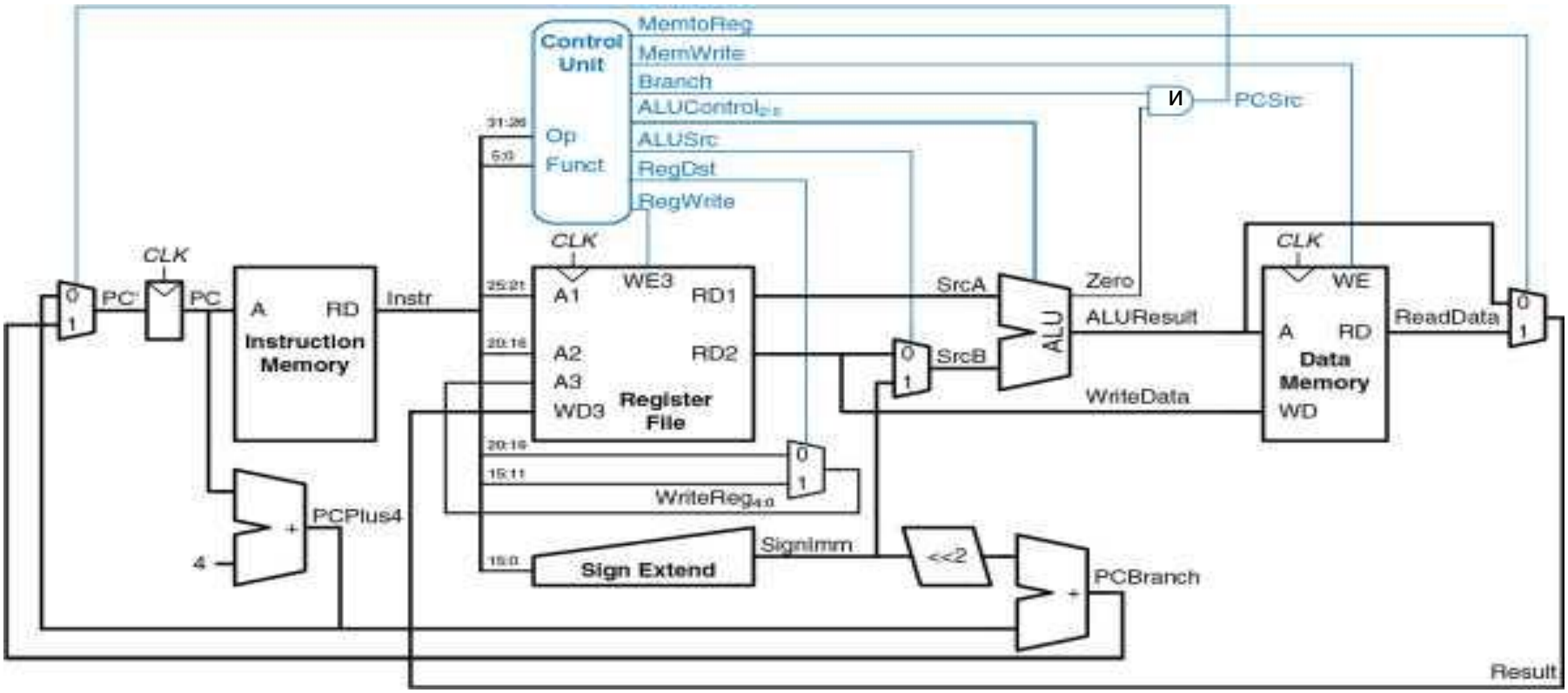


2. **Новое значение счетчика команд** при выполненного условного перехода (**PCBranch**) вычисляется путем сдвига влево расширяющего знака **SignImm** на **два** разряда и последующего сложения с **PCPlus4**. Сдвиг влево на два разряда - *это легкий способ умножения на четыре*, так как сдвиг **влево** на постоянное число разрядов не требует никаких логических элементов, а требует только пере подключения сигналов.

При выполнении условного перехода два регистра сравниваются путем вычитания одного из другого в **АЛУ**. Если **ALUResult** равен **нулю (**т.е**.** сигнал **Zero=1)**, 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,** т.е. **сигнал Branch** равен **единице** для команды **beq** и тогда схема **И** выдает сигнал **PCSrc** для мультиплексора счетчика . Для команды **beq** сигналы **ALUControl = 110**, что означает, что АЛУ должно выполнить операцию вычитания. Если **ALUSrc = 0**, операнд **SrcB** был прочитан из регистрового файла **RD2**. Сигналы управления **RegWrite** и **MEMWrite** равны **нулю**, так как команда условного перехода ничего не пишет ни в **регистровый файл**, ни в **память**. Значения **RegDst** и **MemoReg** **не активны** в регистровый файл также ничего не пишется.

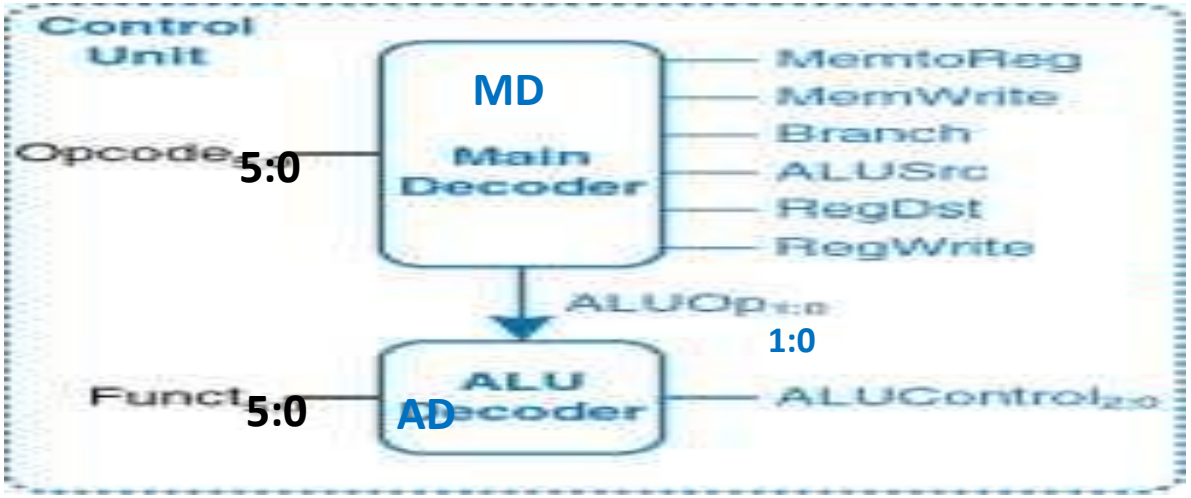
4. Однотактное устройство управления.

Схема однотактного процессора MIPS с устройством управления.



Устройство управления формирует управляющие сигналы на основе полей **opcode** (оперативный код Instr 31:26) и **funct** (функция Instr 5:0), присутствующих в командах. Большая часть информации для устройства управления берется из поля **opcode**, но команды типа **R** также используют и поле **funct** для определения операций в АЛУ.

Устройство управления можно разделить на две части:
комбинационную логику (**Main Decoder - MD**);
дешифратор этой логики (**ALU Decoder -AD**) .



Блок управления.

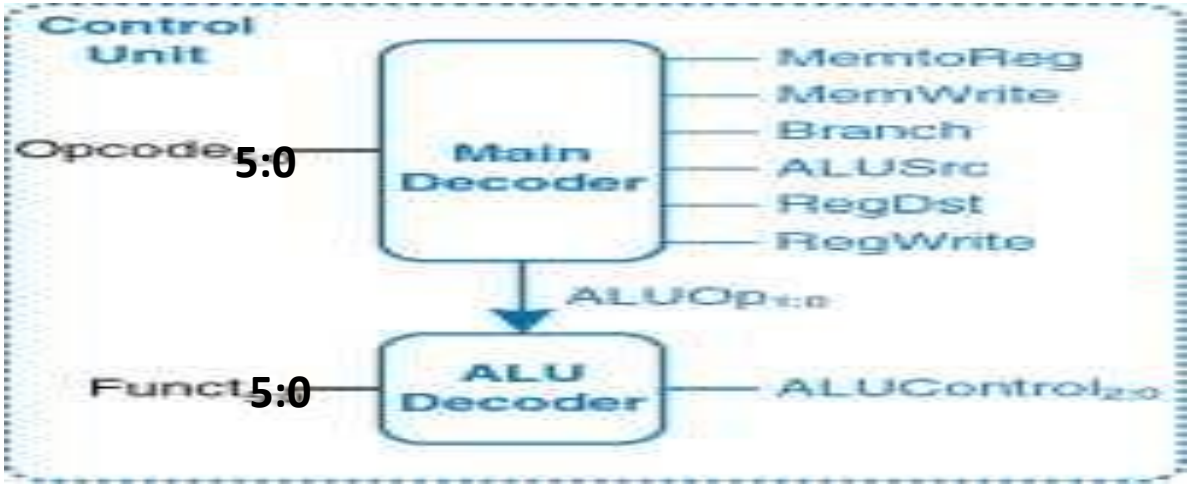
Основной дешифратор **MD** вычисляет значение большинства выходов на основе 6- битного поля **opcode**. Он также формирует **двухбитный** сигнал **ALUOp (1:0)** для дешифратора логики **AD** . Дешифратор АЛУ (**AD**) использует 2-х битный сигнал **ALUOp** совместно с 6- битным полем **Opcode** для вычисления 3-х битного состояния **ALUControl (2:0).**

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

Управляющие сигналы всех команд **главного дешифратора MD**: MemotoReg, MemWrite, Branch, ALUSrc,RegDst, RegWrite уже были описаны при анализе тракта данных.

Так как сигнал **ALUOp** =**11** никогда не используется, то для упрощения логики можно использовать неопределенные значения X1 и 1X вместо 01 и 10,т.е. контролировать только место **1** в старшем или младшем бите управления.

Для команд типа **R** первые два бита поля **funct** всегда равны **10**, так что их можно проигнорировать для упрощения дешифратора.

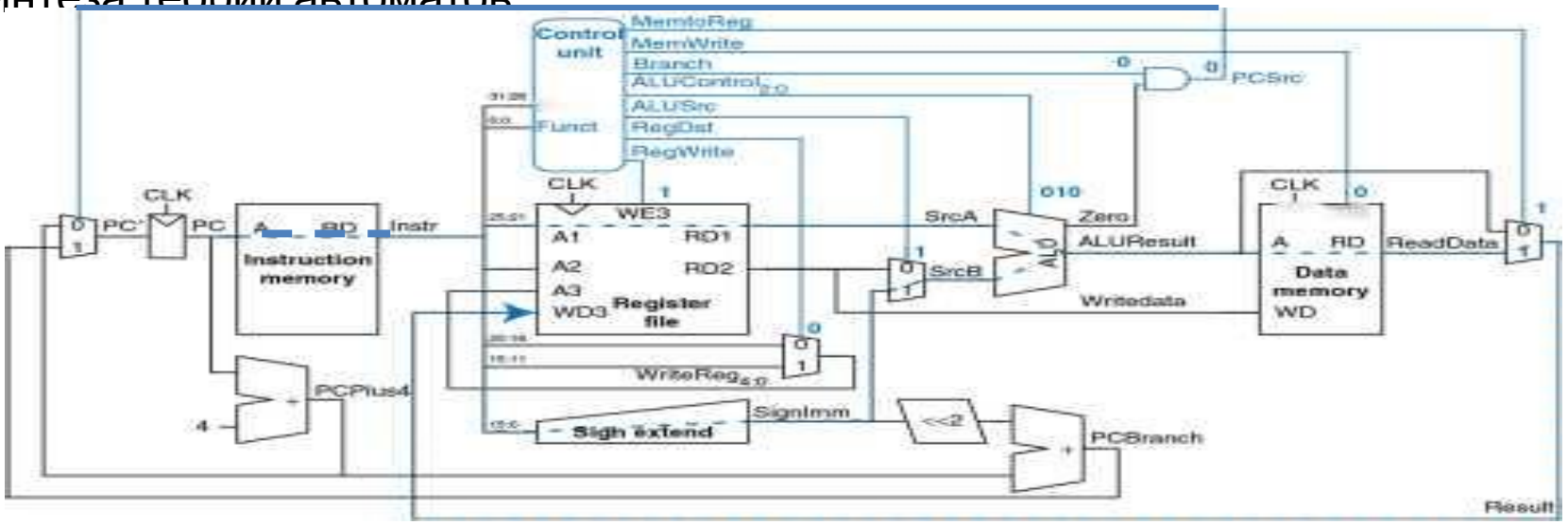


ALUOp(0:1)	Opcode 6b	ALUControl (2:0)
00	X	010 (сложение)
X1	X	110 (вычитание)
1X	100000 (add)	010 (сложение)
1X	100010 (sub)	110 (вычитание)
1X	100100 (and)	000 (логическое «И»)
1X	100101 (or)	001 (логическое «ИЛИ»)
1X	101010 (slt)	111 (установить, если меньше)

Таблица истинности для основного дешифратора, показывающая зависимость управляющих сигналов от значения составляющей **opcode**

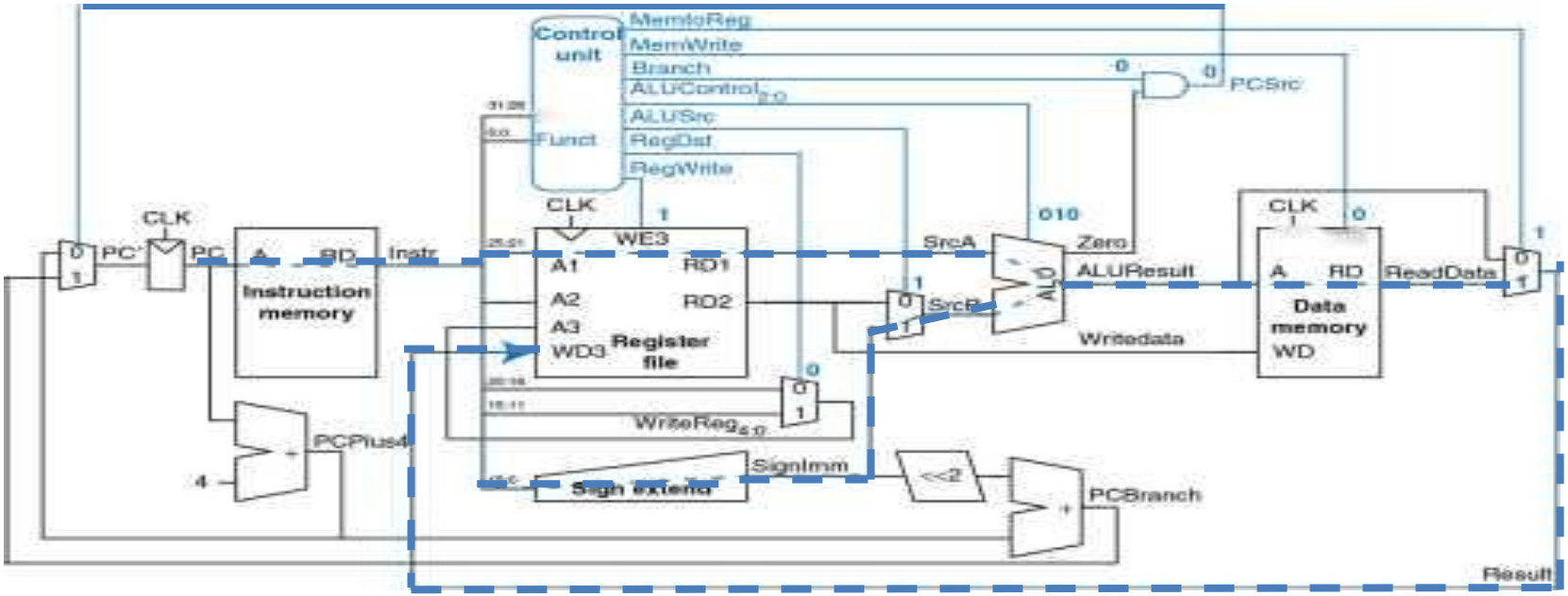
Дешифратор

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 **WD3** файла регистров **RF**, не имеют никакого значения, так как RegWrite равен **нулю**. Для создания дешифратора можно использовать любой известный метод синтеза теории автоматов



Команда	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp
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

5. Анализ производительности однотактного процессора.



Каждая команда в однотактном процессоре выполняется ровно за один такт.

Цепь с наибольшей задержкой для команды **lw** показана **синей пунктирной линией**.

Начало в **счетчике команд** по положительному фронту тактового сигнала записывает свое новое значение. Затем **обновленное значение PC** используется для выборки следующей команды. Потом процессор читает **SrcA** из регистрового файла, **одновременно** с этим **читается знаковое расширенное** смещение **Sign**, которое через мультиплексор **SrcB** подается на вход **АЛУ** как операнд. АЛУ **складывает операнды SrcA и SrcB**, вычисляя эффективный адрес памяти **ALUResalt** . По этому адресу **производится чтение из памяти данных ReadData**, которое выбирается мультиплексором. **Получаем сигнал Result**. Сигнал **Result** должен **стать стабильным** на входе регистрового файла до того, как придет следующий положительный фронт тактового сигнала, иначе запись будет иметь неверное значение.

Минимальную длительность одного такта можно подсчитать следующим образом:

$$T_c = t_{pcq_PC} + t_{mem} + \max[t_{RFread},] + t_{ALU} + t_{mem} + t_{mux} + t_{RFsetup}$$

где:

t_{pcq_PC} – переключение счетчика программ PC;

t_{mem} - время чтения из памяти инструкций instruction memory;

t_{RFread} - время чтения регистрового файла RF;

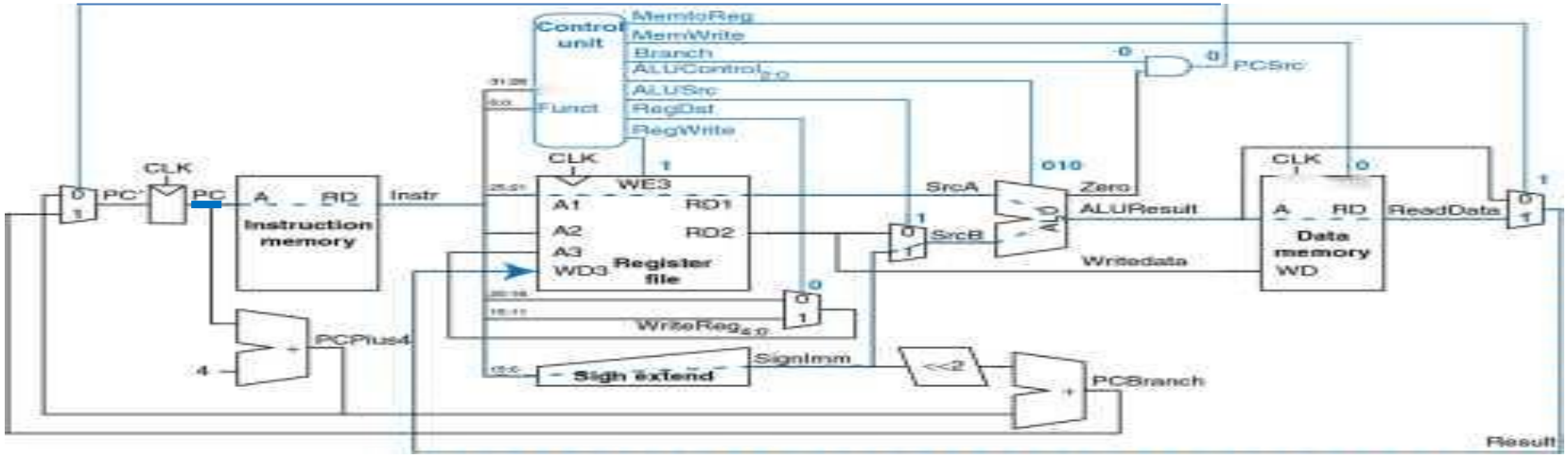
t_{ALU} – время обработки операций в АЛУ;

t_{mux} – время работы мультиплексора;

$t_{RFsetup}$ – время установки регистрового файла WD Reg write.

В большинстве технологий производства микросхем **доступ к АЛУ, памяти и регистровым файлам** занимает гораздо больше времени, чем прочие операции и тогда длительность одного такта приближенно можно оценить следующим образом:

$$T_c = t_{pcq_PC} + 2t_{mem} + t_{RFread} + t_{ALU} + t_{mux} + t_{RFsetup}$$

С учетом средних значений работы элементов имеем:

T_c= **30 + 2(250) + 150 + 200 + 25 + 20 = 925 нс**.