Структурная и функциональная организация ЭВМ (Computer Organization and Design)

БГУИР
кафедра ЭВМ

Лекция 11
«Проектирование однотактного ЦП»

План лекции

1. Форматы команд различных процессоров
2. Проектирование процессоров основываясь на наборе команд и их формате
3. Однотактный ЦП

Ключевые моменты АСК (ISA)

- длина команд
  - все команды одной длины или нет?
- сколько регистров?
- где могут находиться операнды?
  - т.е., можно ли сложить напрямую содержимое ячейки памяти и регистра?
- формат команды
  - какие биты - что определяют?
- операнды
  - сколько? какой длины?
  - каким образом вычислять исполнительные адреса памяти?
- операции
  - какие операции над данными совершает ВМ?

Формат команды

– что каждый бит команды означает?

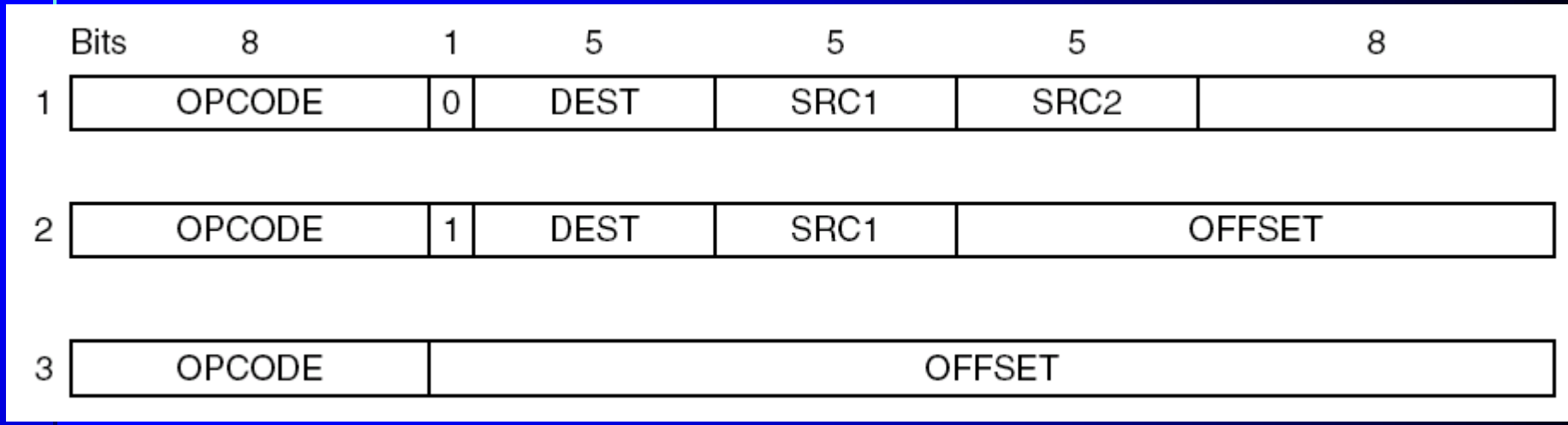
ВМ должна уметь быстро определять,

- "Это 6-байтная команда"
- "Виты 7-11 определяют регистр"
- ...
- Последовательное декодирование – плохой вариант

В случае если есть много различных вариантов форматов команд...

- усложнение процесса декодирования команды
- необходимость использования бит команды для кодирования формата команды

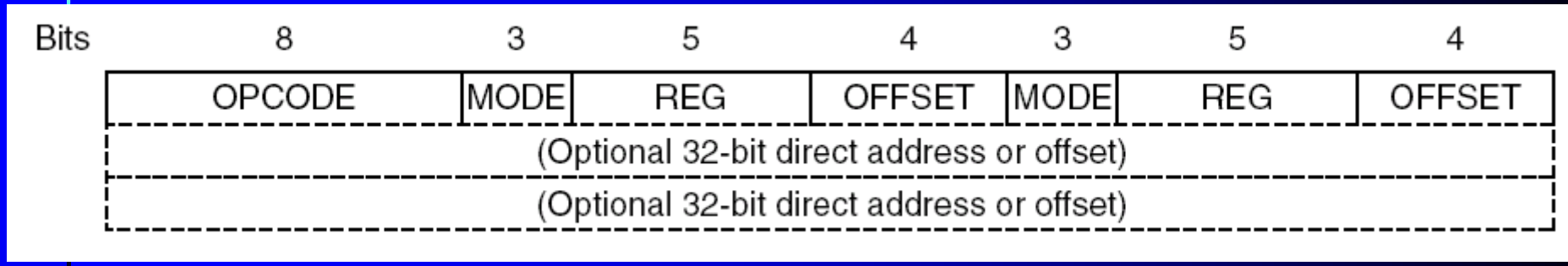
Ортогональность КОП и способов адресации



Простой дизайн формата команды для ВМ с тремя операндами. Идеально – регулярная структура формата команд и их число должно ->min.

Все регистры должны быть доступны для всех регистров, включая FP (указатель фрейма), SP (указатель стека) и PC (счётчик команд).

Ортогональность КОП и способов адресации



Простой дизайн формата команды для ВМ с двумя операндами. ВМ должна прибавлять слово из памяти к регистру, регистр к слову, складывать два регистра либо два слова из памяти.

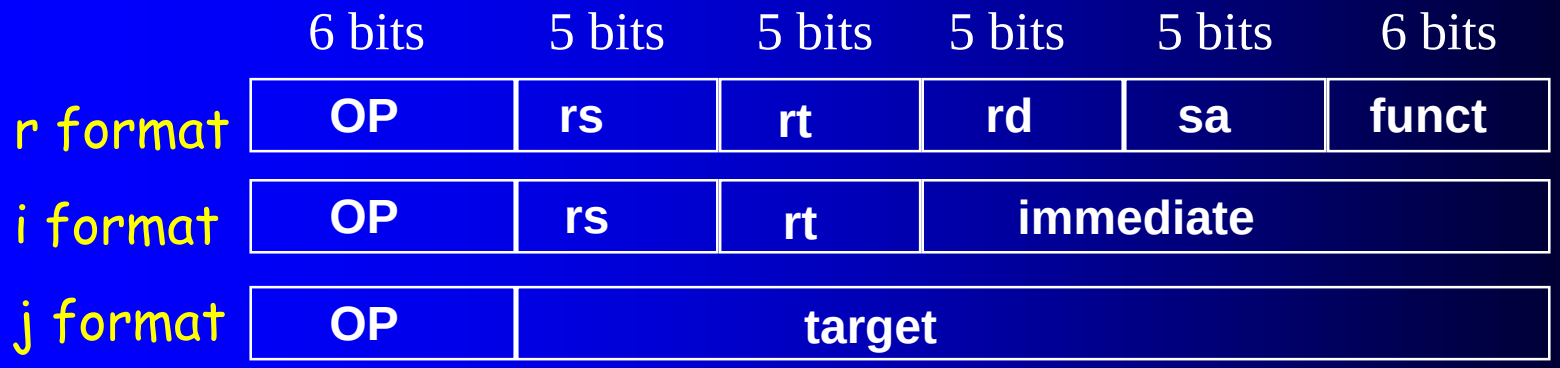
Пока доступ к памяти дорог – этот подход не популярен.

PDP-11 и VAX – сходные системы, были популярны.

Проблема – при прямой адресации – большое кол. бит для адреса (96) -> 3 цикла шины (команда +2 операнда).

Для любого RISC – тоже 96 бит команда + 4 цикла.

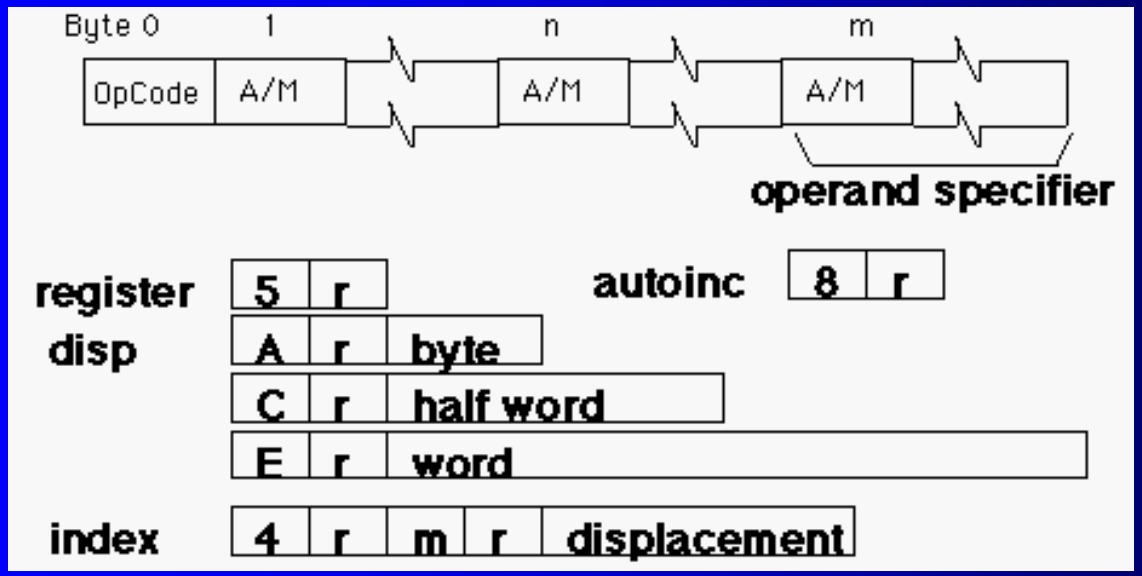
MIPS Instruction Formats



- например, "`add r1, r2, r3`" будет
  - OP=0, rs=2, rt=3, rd=1, sa=0, funct=32
  - 000000 00010 00011 00001 00000 100000
- код операции (КОП - OP) автоматически задаёт и формат команды

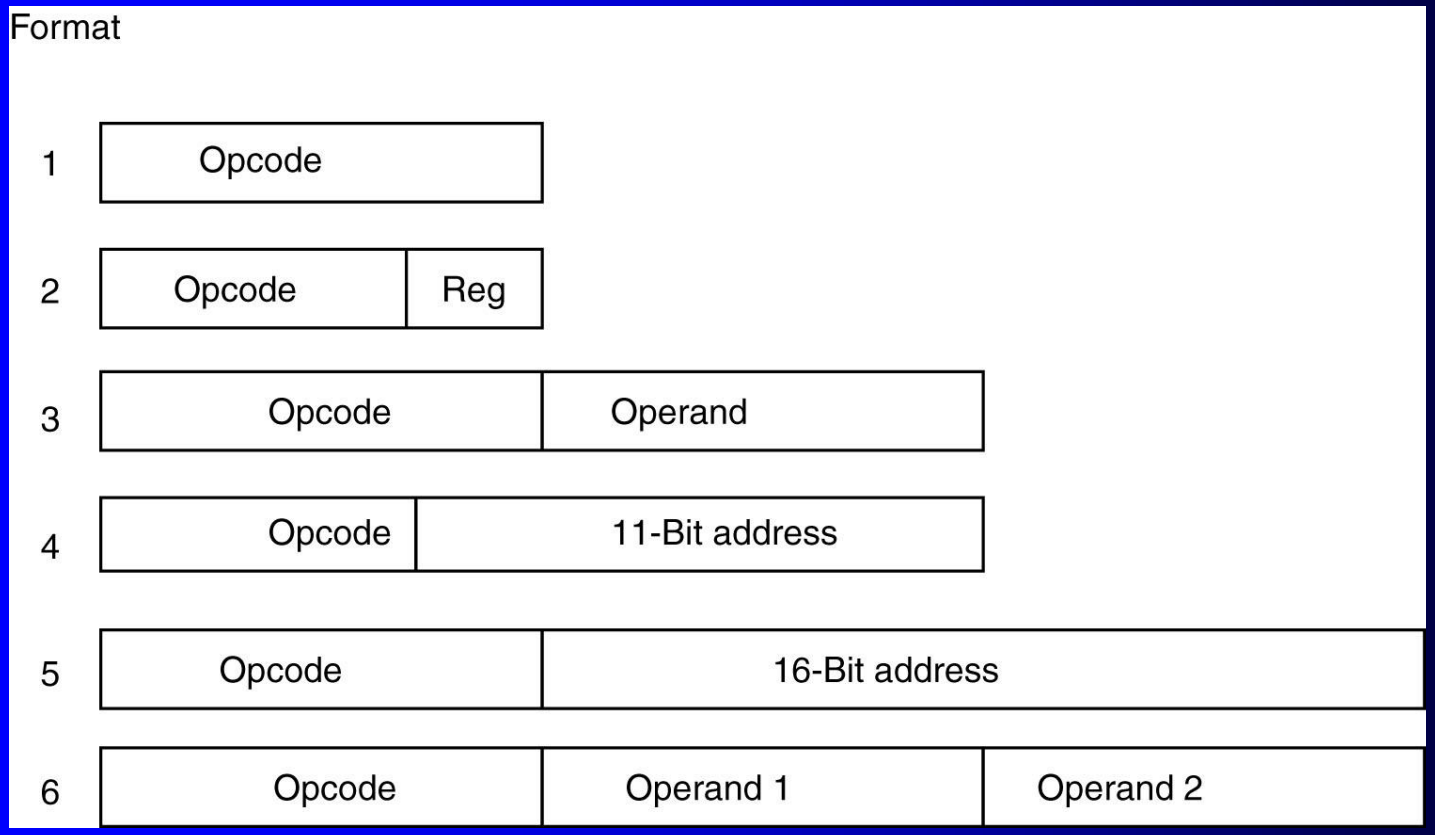
VAX Instruction Formats

- 1 байтный КОП (OPcode)
  - определяет количество, тип и длину операндов
- Каждый операнд от 1 до нескольких байт
  - первый байт определяет способ адресации

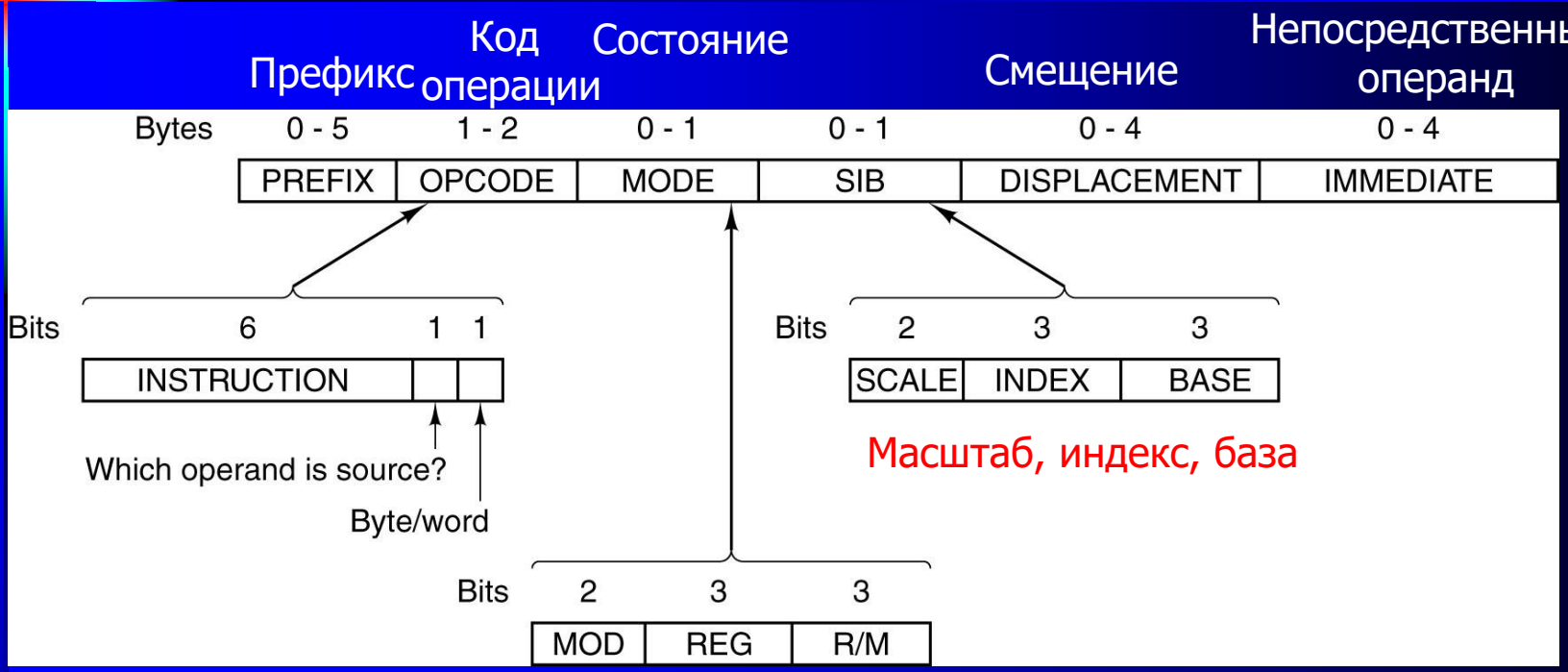


“move” can be load, store, mem copy, jump,... depending on operands

The 8051 Instruction Formats

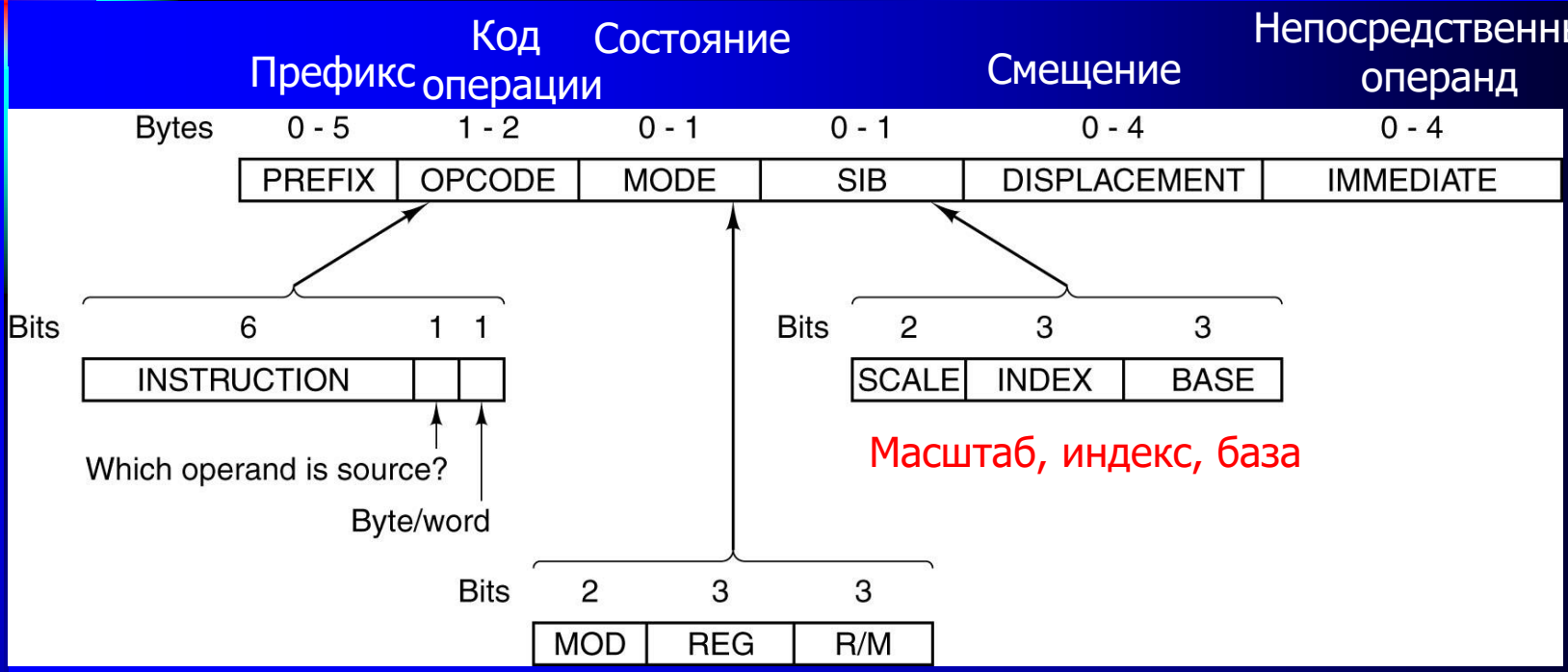


Формат команд Pentium II - Pentium 4



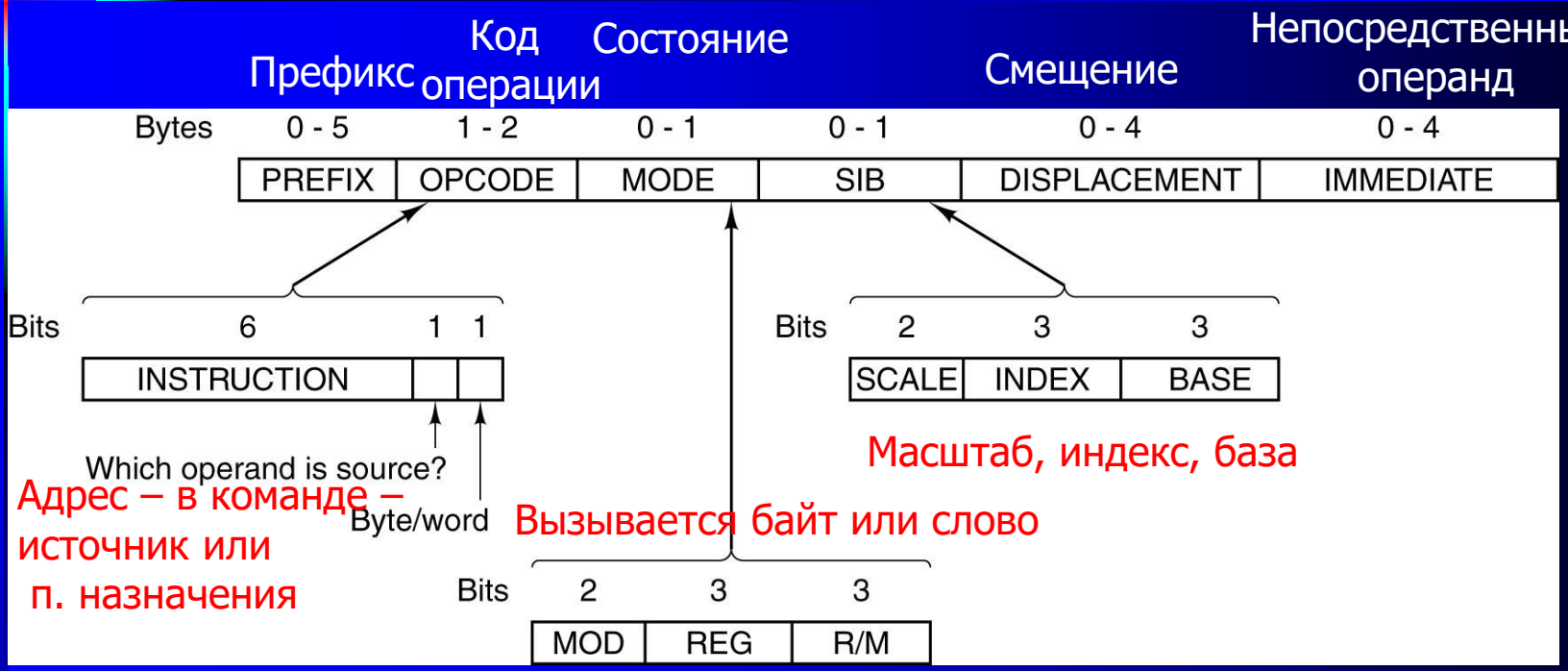
В первых Intel КОП (OpCode) – 1 байт.
Префиксный байт – доп. код операции, который ставится перед командой, для изменения её действия.
0xFF – код смены алфавита – разрешает второй байт команды.

Формат команд Pentium II - Pentium 4



Основной минус – команда должна быть сперва полностью декодирована, чтобы выяснить к какому классу относится операция -> нельзя до декодирования определить где начинается следующая команда.

Формат команд Pentium II - Pentium 4



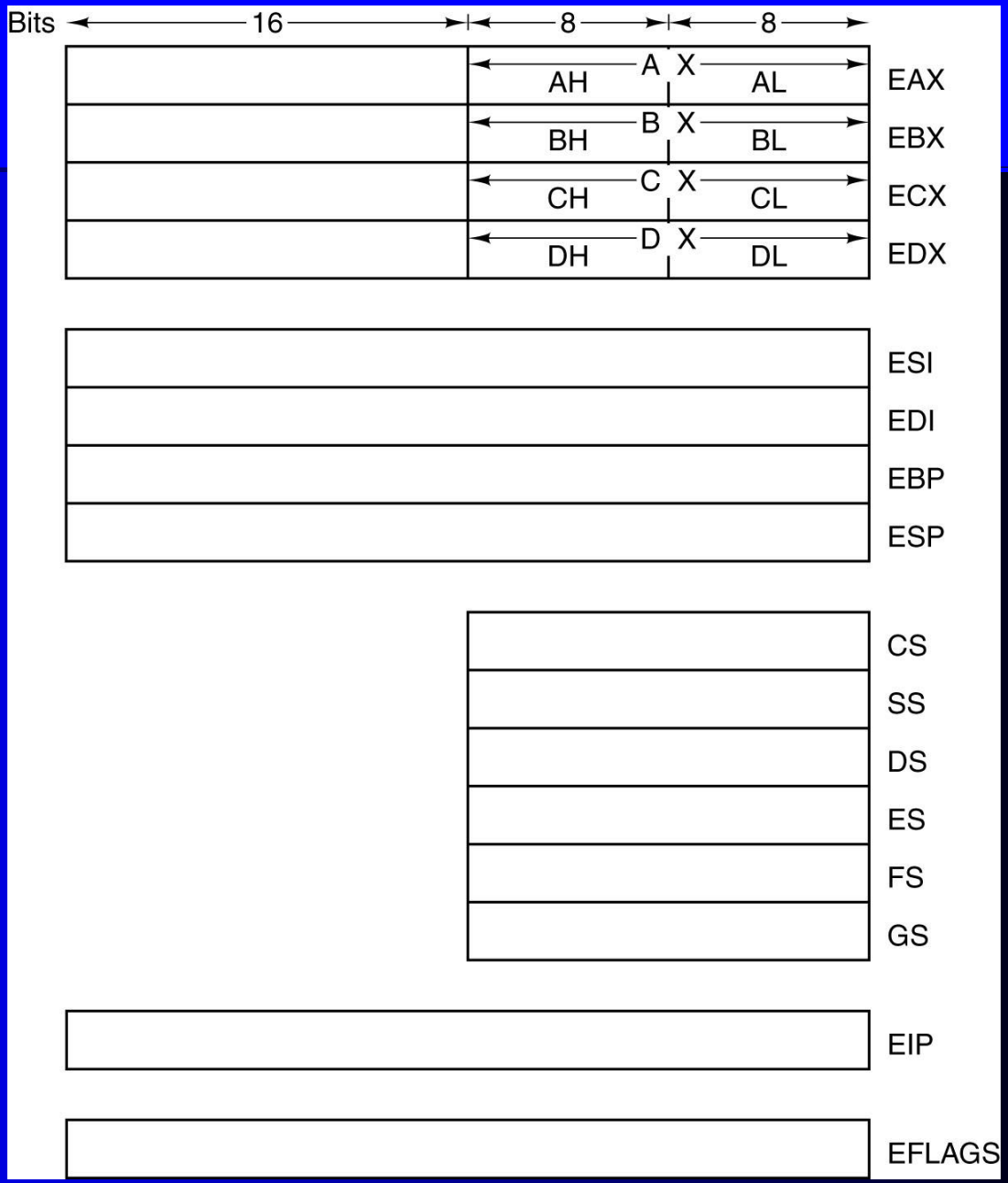
Часто за КОП следует байт обобщения об операнде.
R/M – часто исп. для расширения КОП. Mode – 2 бита = 4 способа обращения к операндам, один из них всегда регистр.
REG – комбинации регистров EAX,EBX,ECX,EDX,ESI,EDI,EBP,ESP.
В некоторых командах – доп. байт SIB – доп. спецификация (компромисс м/у обратной совм. и добавл. новых возможностей).

The Pentium 4 Addressing Modes

	MOD			
R/M	**00**	**01**	**10**	**11**
000	M[EAX]	M[EAX + OFFSET8]	M[EAX + OFFSET32]	EAX or AL
001	M[ECX]	M[ECX + OFFSET8]	M[ECX + OFFSET32]	ECX or CL
010	M[EDX]	M[EDX + OFFSET8]	M[EDX + OFFSET32]	EDX or DL
011	M[EBX]	M[EBX + OFFSET8]	M[EBX + OFFSET32]	EBX or BL
100	SIB	SIB with OFFSET8	SIB with OFFSET32	ESP or AH
101	Direct	M[EBP + OFFSET8]	M[EBP + OFFSET32]	EBP or CH
110	M[ESI]	M[ESI + OFFSET8]	M[ESI + OFFSET32]	ESI or DH
111	M[EDI]	M[EDI + OFFSET8]	M[EDI + OFFSET32]	EDI or BH

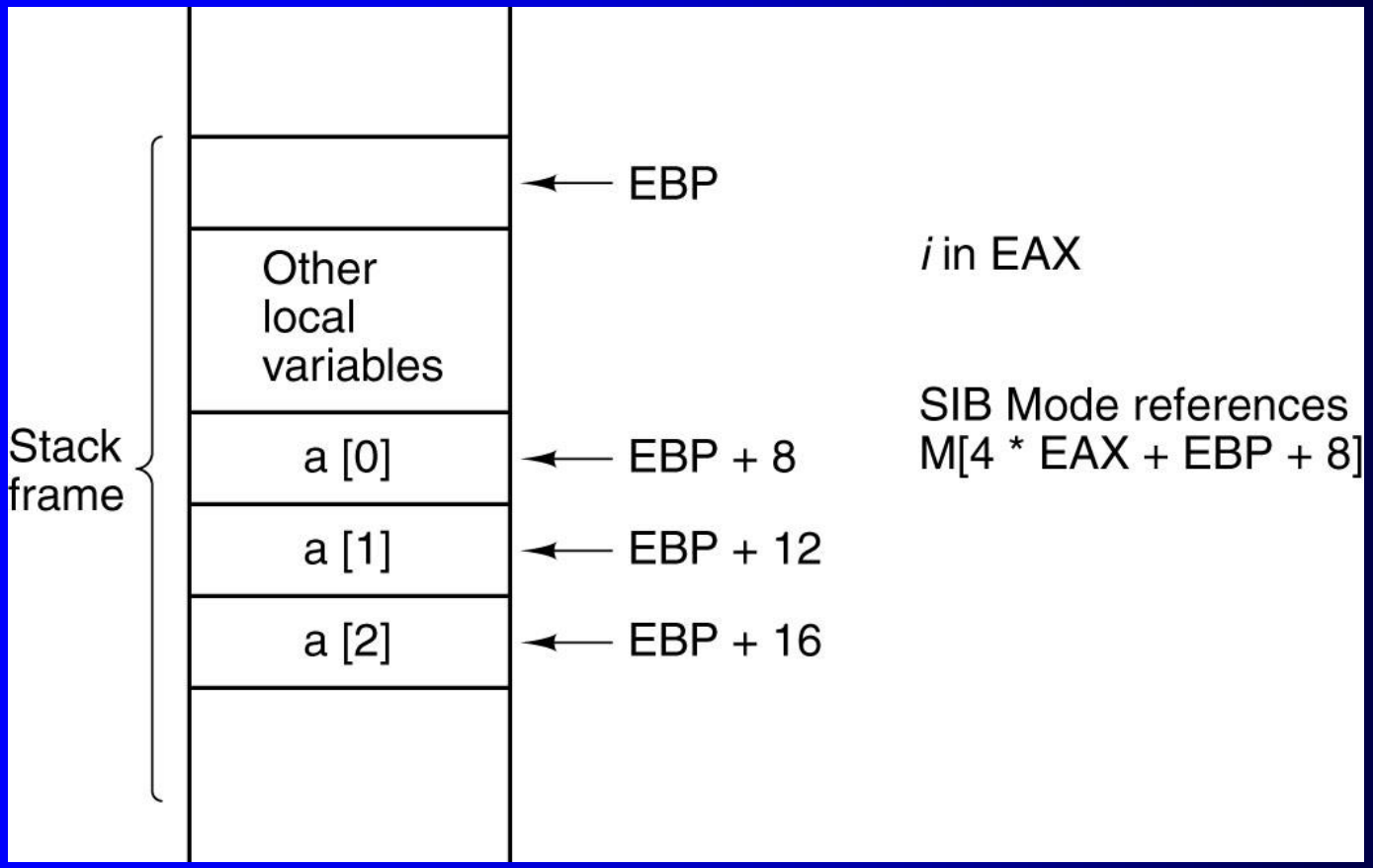
M[*x*] is the memory word at *x*.

Основные регистры Pentium 4



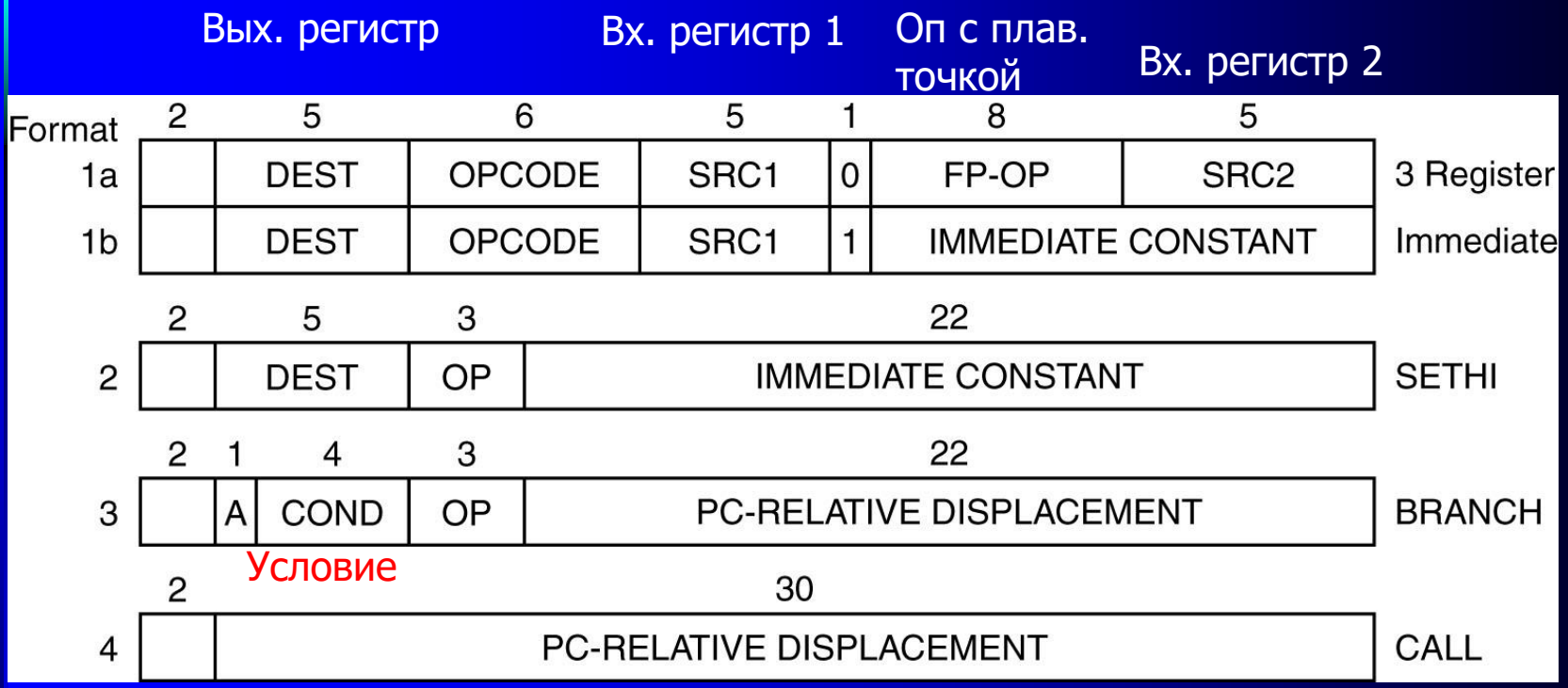
The Pentium 4 Addressing Modes

Доступ к *a*[*i*].



The UltraSPARC III Instruction Formats

The original SPARC instruction formats.



Смещение относительно счетчика команд

Для включения в 32-битную команду 32 битной конст. – SETHI команда – задание в два этапа (22 бита + 10 бит следующей командой)

Discussion of Addressing Modes

Сравнение способов адресации.

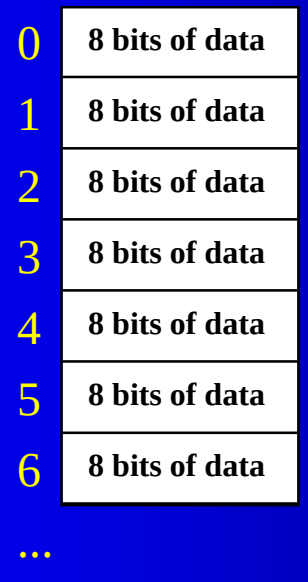
Addressing mode	Pentium 4	UltraSPARC III	8051
Accumulator			×
Immediate	×	×	×
Direct	×		×
Register	×	×	×
Register indirect	×	×	×
Indexed	×	×	
Based-indexed		×	
Stack			

Количество операндов?

- Двух-операндный (two-address) код: место для хранения результата – совпадает с местом хранения операнда (не всегда удобно):
  - x = x + y
- Трёх-операндный (three-address) код: место для хранения результата может отличаться
  - x = y + z
  - x86 не имеет трёх-операндных команд; у других встречаются
- Некоторые операнды задаются по умолчанию
  - "условный код" производит действия в зависимости от значений результата +, 0, или –
  - PowerPC имеет команду "**Branch on count**" использующую спец. "count register"
- Известные АСК имеют от 0 до 4 операндов
  - PowerPC команда - "**float multiply add**", r = x + y*z
  - «It also has fancy bit-manipulation instructions»

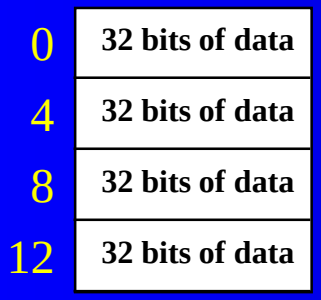
Пара слов об адресации памяти

- Рассматривается как большой одномерный массив ячеек с последовательными адресами.
- Адрес – индекс массива
- Побайтовая адресация ("Byte addressing") означает что индекс указывает на какой-то один байт памяти.



Пара слов об адресации памяти

- Байты хорошо – но большинство данных требует большее количество бит одновременно - «слов»
- Для MIPS слово – 32 бита или 4 байта.



Регистры содержат 32 бита данных

- 2^{32} bytes with byte addresses from 0 to $2^{32}-1$
- 2^{30} words with byte addresses 0, 4, 8, ... $2^{32}-4$
- Words are aligned
  i.e., what are the least 2 significant bits of a word address?

АСК MIPS

- фиксированные 32-битные команды
- 3 типа формата команд
- 3-операндная архитектура «load-store»
- 32 РОН (integer, floating point)
  - R0 всегда равен 0.
- 2 целочисленных регистра спец. назначения, HI и LO, (умножение и деление производят результат, занимающий больше, чем 32 бита).
- 32-х разрядные регистры (word)
- Регистровая, непосредственная, и базовая со смещением адресация

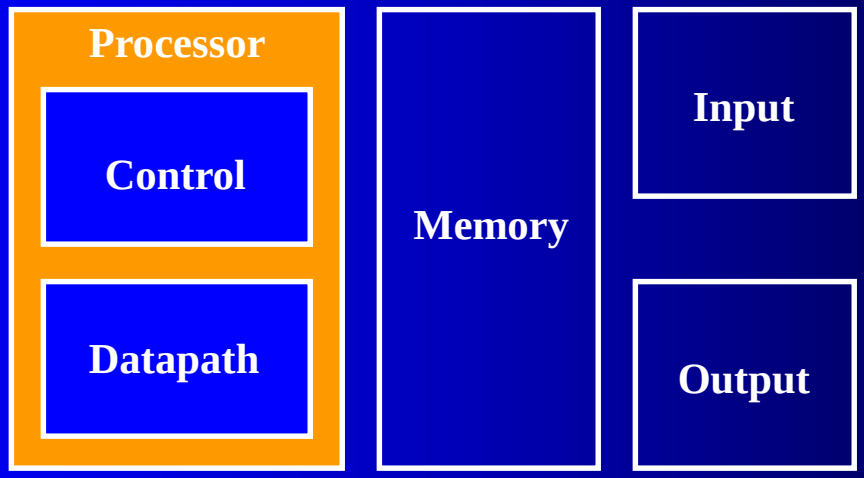
Key Points

- Четыре принципа разработки АСК
  - регулярность порождает простоту (regularity produces simplicity)
  - чем меньше - тем быстрее (smaller is faster)
  - хорошее решение всегда в компромиссе (good design demands compromise)
  - общие вещи должны выполняться как можно быстрее (make the common case fast)

Проектирование однотактного ЦП
Single Cycle CPU Design

Что требуется

- Пять классических блоков ЭВМ



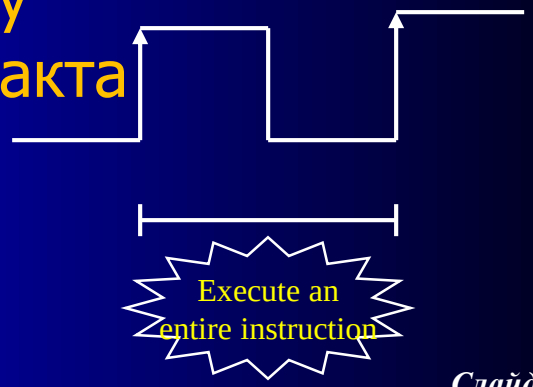
Производительность

- Время выполнения - ET

  ET = Insts * CPI * Cycle Time

- Конструкция процессора (тракта данных и УУ) определяет:
  - Время тактового импульса
  - Количество импульсов для выполнения одной команды

  - Однотактовый процессор:
    - Плюсы: Один такт на команду
    - Минусы: Длительное время такта



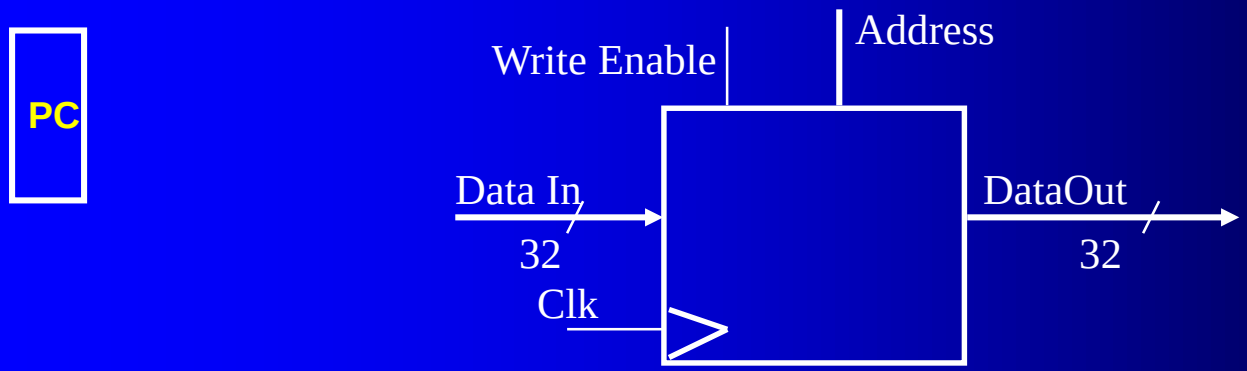
The Processor: Datapath & Control

- Рассмотрим упрощённую версию MIPS, содержащую только:
  - команды работы с памятью: **`lw, sw`**
  - арифметико-логические команды: **`add, sub, and, or, slt`**
  - команды управления потоком команд: **`beq`**

- Общий цикл:
  - Использование счётчика команд СК - the program counter (PC) для определения адреса комады
  - Получение команды из памяти
  - чтение регистров
  - использование КОП для определения конкретных действий

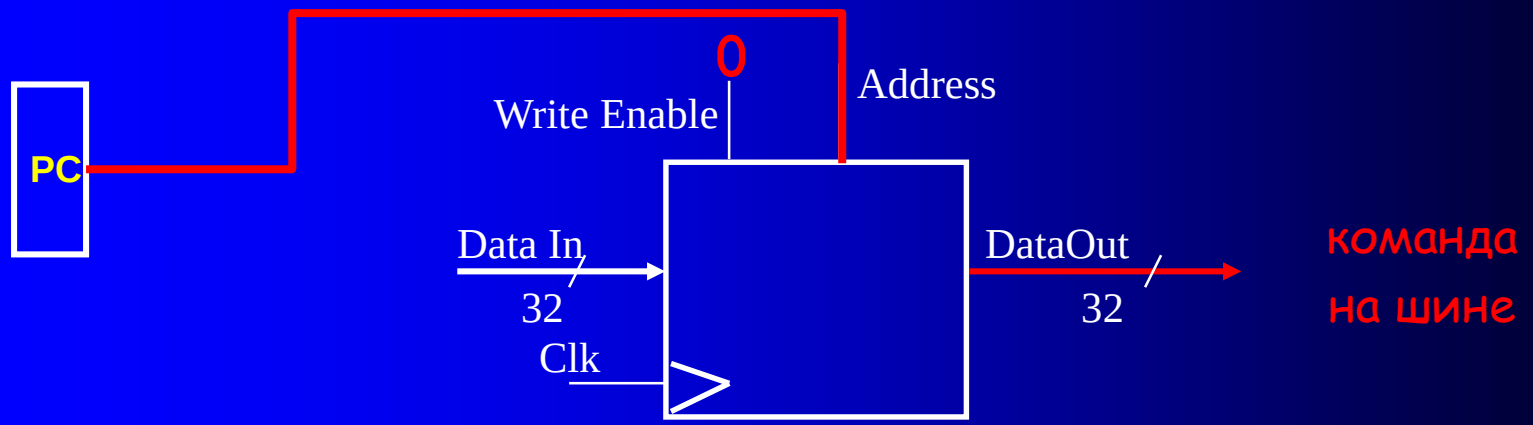
Processor Design

- Во-первых, нам необходимо загрузить команду из памяти в процессор
  - Счётчик команд - program counter (PC) отвечает за адрес команды
  - Получение команды обеспечивается памятью



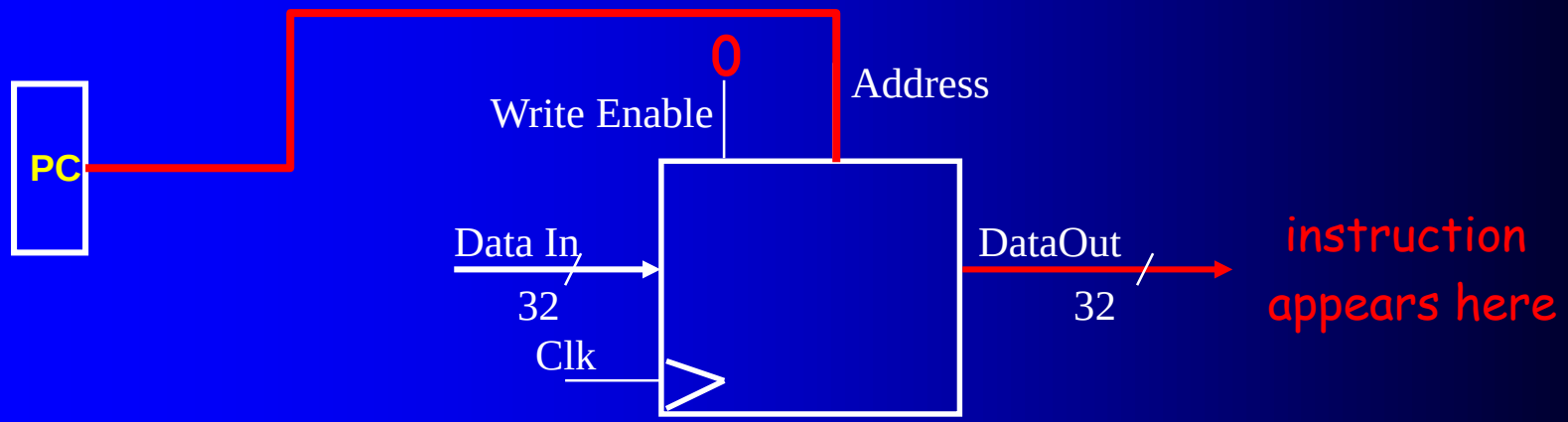
Processor Design

- Во-первых, нам необходимо загрузить команду из памяти в процессор
  - Счётчик команд - program counter (PC) отвечает за адрес команды
  - Получение команды обеспечивается памятью



That was too easy

- Проблема – как мы будем загружать или сохранять?
  - память имеет только один порт
  - мы хотим сделать всё за один цикл

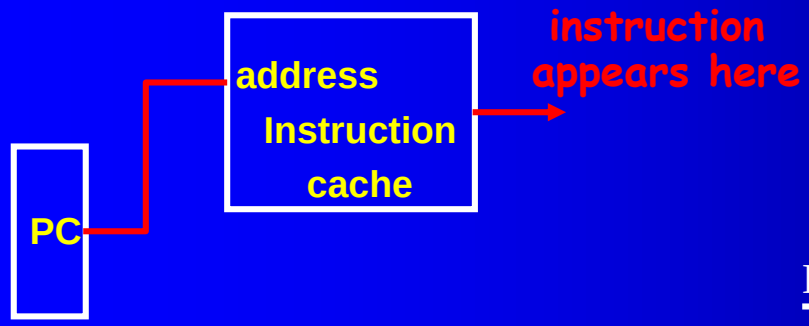


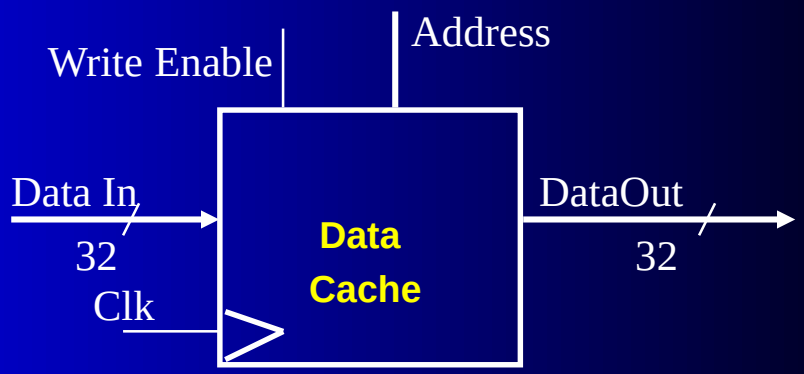
Instruction & Data in same cycle?

Решение: отдельные блоки памяти для данных и команд

- Будет только одна DRAM память
- Но можно включить раздельные блоки SRAM кэш-памяти

(Будем рассматривать позже)

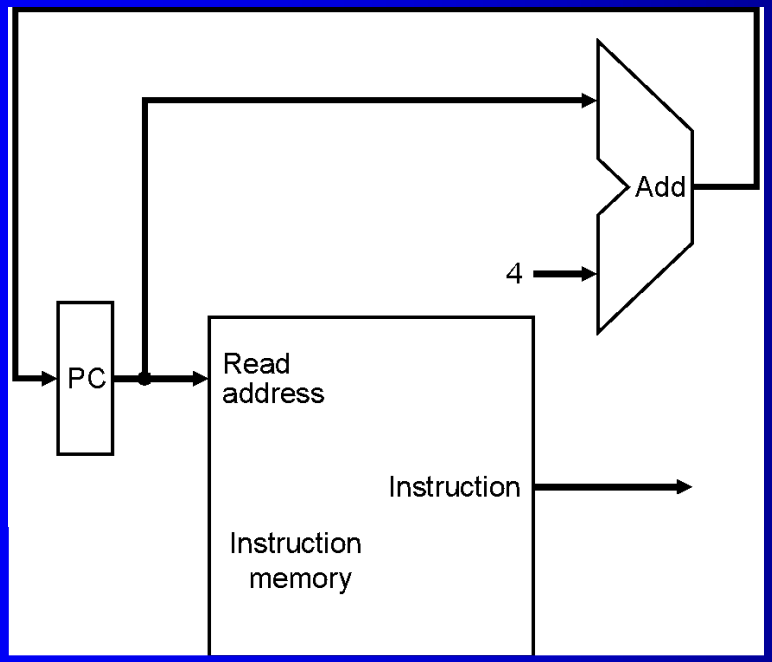




Instruction Fetch Unit

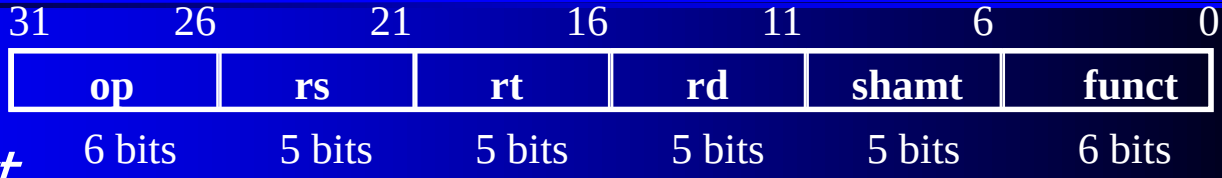
Обновление СК (PC) для следующей команды

- В случае последовательного перехода: PC <- PC + 4
- Ветвление и Jump: PC <- “что-то другое”
  - Позаботимся об этом позже



The MIPS core subset

- R-type
  - *add rd, rs, rt*
  - sub, and, or, slt



1. Прочитать регистры rs и rt
2. Загрузить их в АЛУ
3. Обновить содержимое пула регистров

- LOAD and STORE
  - lw rt, rs, imm
  - sw rt, rs, imm

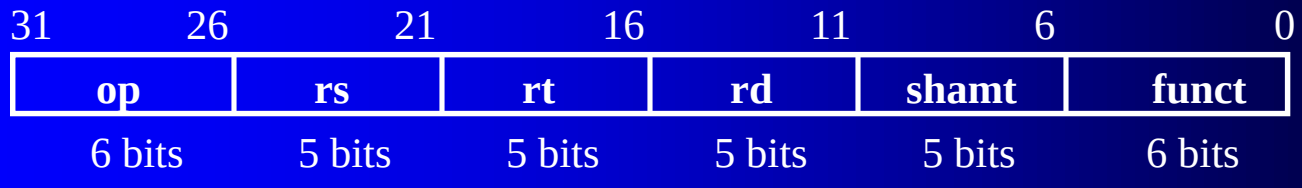
1. Прочитать регистр rs (и rt если store)
2. Загрузить rs и смещение в АЛУ
3. Переместить данные между mem и reg

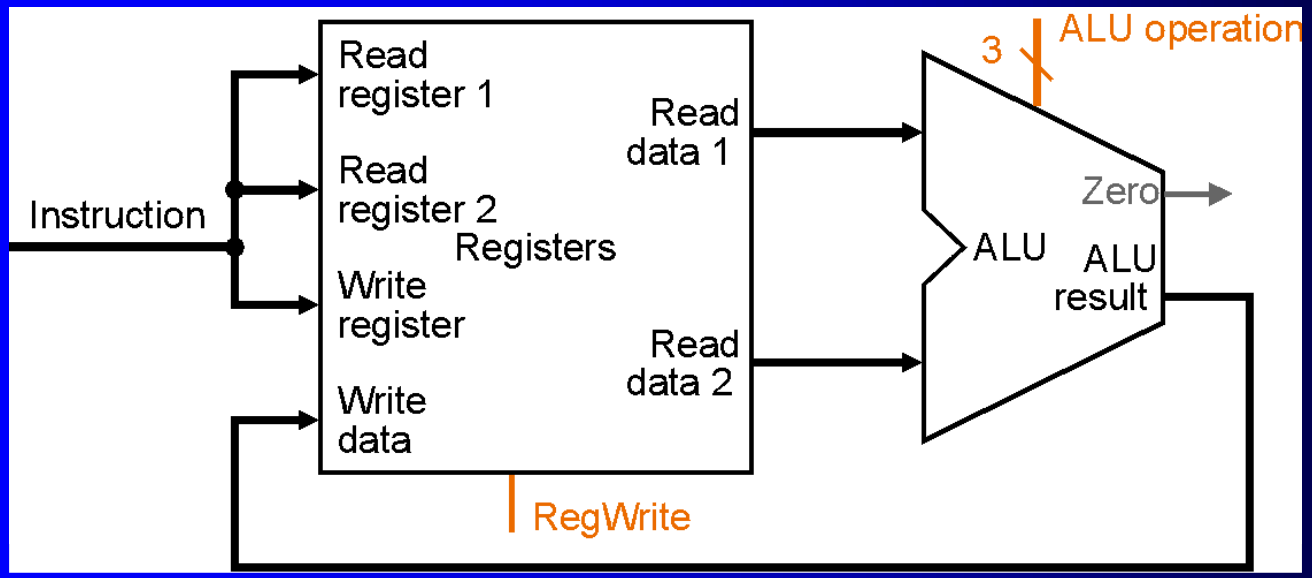
- BRANCH:
  - beq rs, rt, imm

1. Прочитать регистры rs и rt
2. Загрузить в ALU для сравнения
3. Добавить в PC смещение; обновить PC

Datapath for Reg-Reg Operations

- R[rd] <- R[rs] op R[rt] Пример: *add rd, rs, rt*
  - Ra, Rb, and Rw берём из полей rs, rt, и rd
  - Сигналы управления АЛУ (ALUoperation) зависят от КОП команды и определяют функцию АЛУ



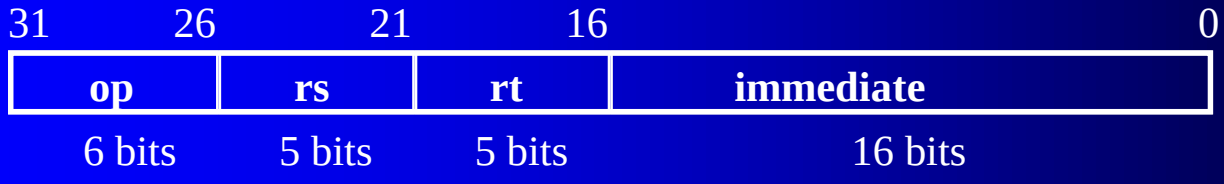


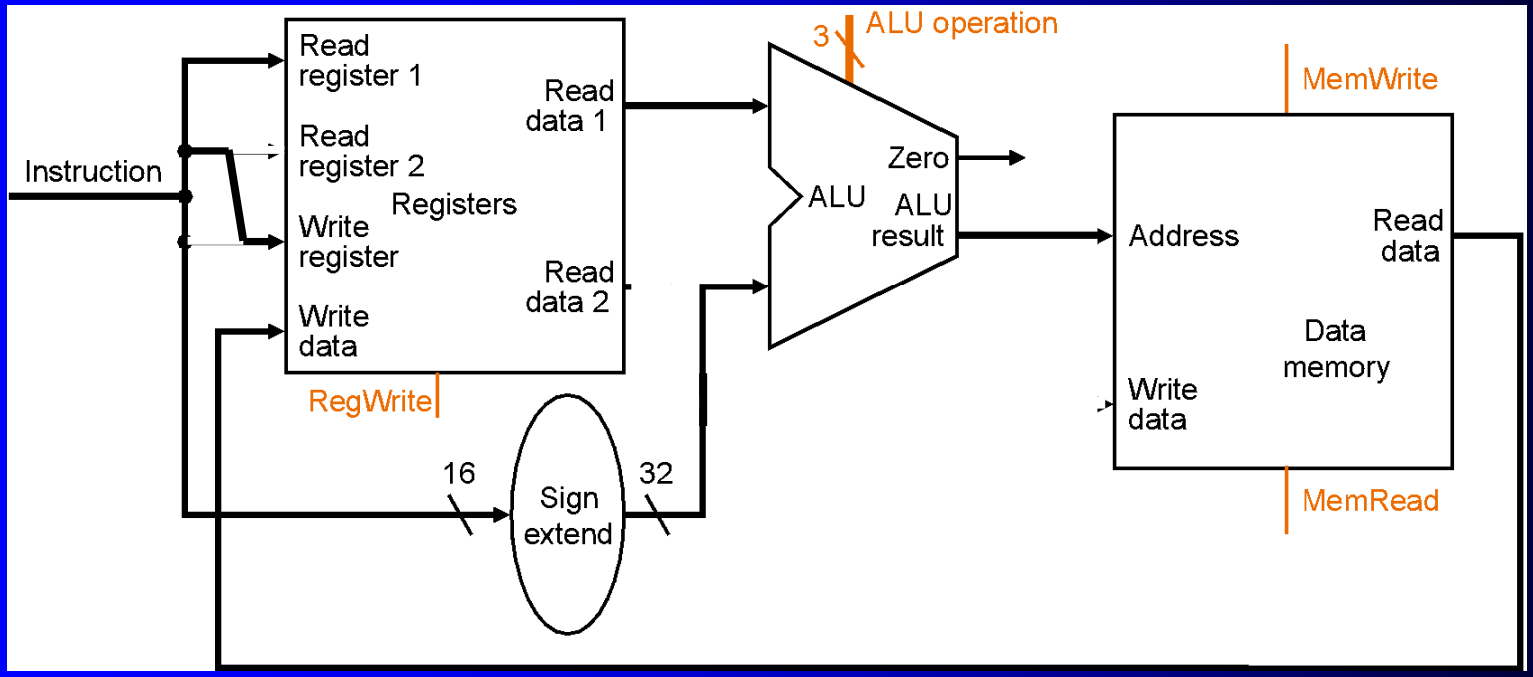
ALUoperation и *RegWrite*: управляют логикой после декодирования команды

Datapath for Load Operations

R[rt] <- Mem[R[rs] + SignExt[imm16]]

Пример: *lw rt, rs, imm16*

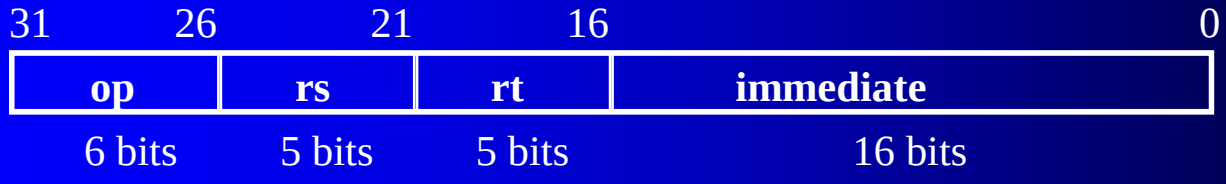


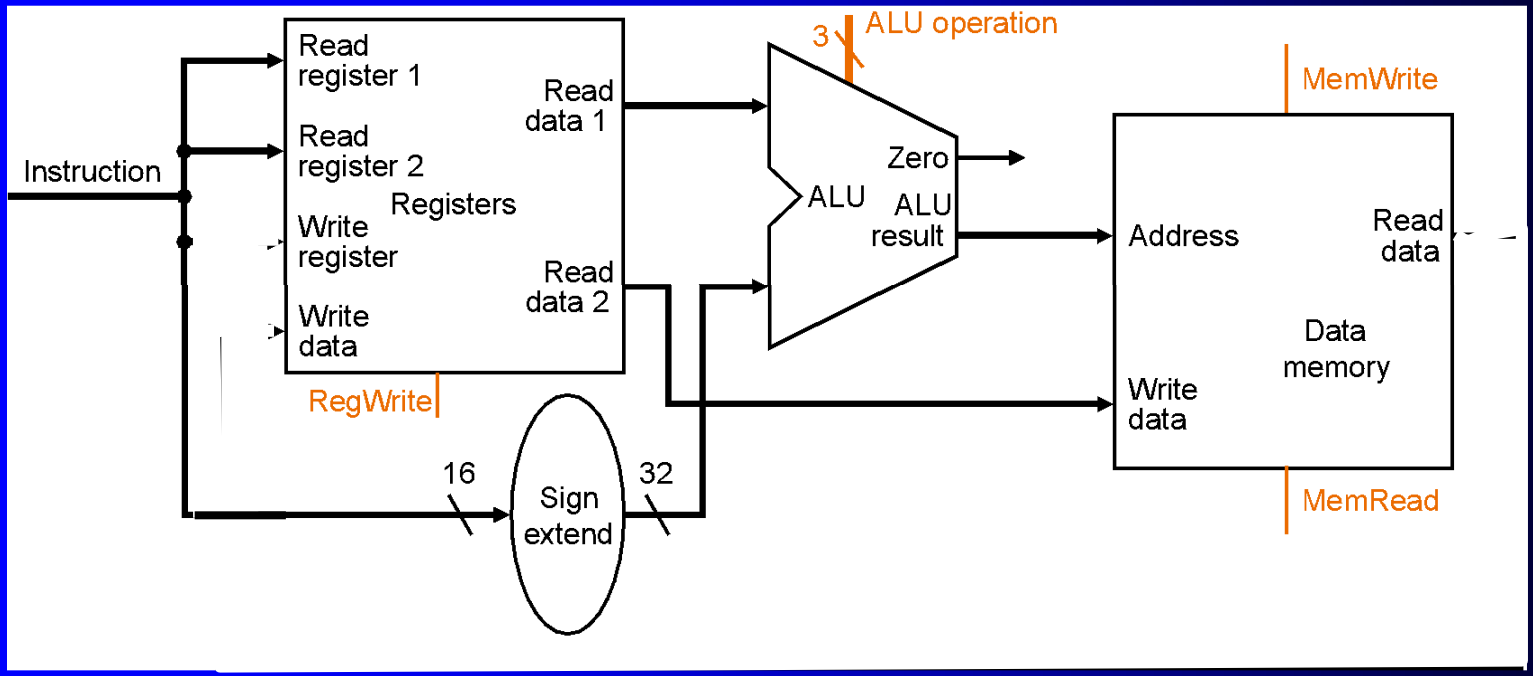


Datapath for Store Operations

Mem[R[rs] + SignExt[imm16]] <- R[rt]

Пример: *sw rt, rs, imm16*

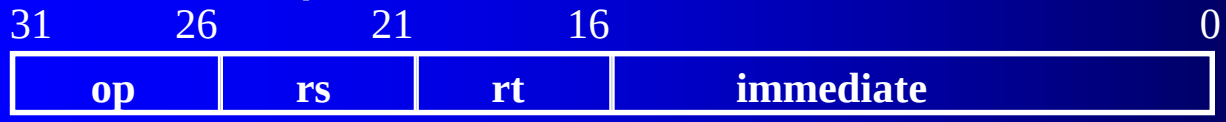


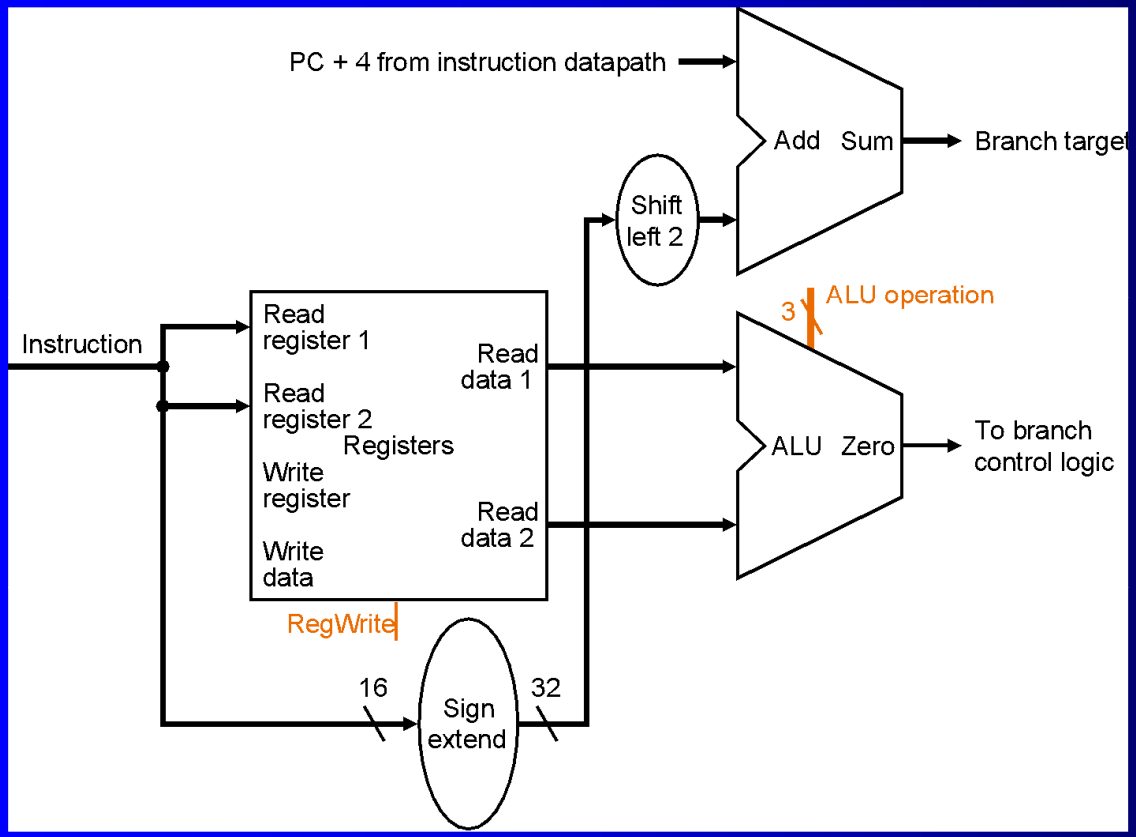


Datapath for Branch Operations

beq rs, rt, imm16

Необходимо сравнить Rs и Rt



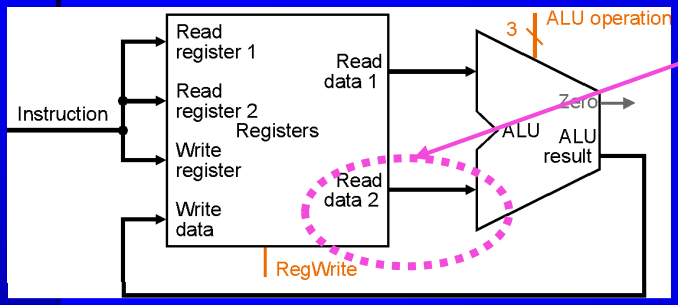


Вычисление следующего адреса

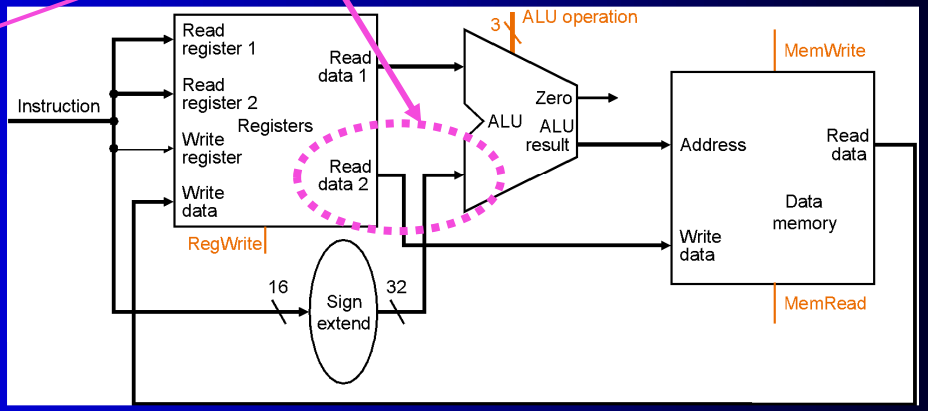
- СК (PC) - 32-bit byte address в памяти команд:
  - Последовательный ход: PC<31:0> = PC<31:0> + 4
  - Ветвление: PC<31:0> = PC<31:0> + 4 + SignExt[Imm16] * 4
- Нам не нужны 2 младших бита так как:
  - 32-bit PC – адресует байт
  - Все наши команды - 4 байта (32 bits) длиной
  - 2 младших бита 32-bit PC всегда нули
  - На практике можно упростить железо используя 30-bit PC <31:2>:
    - Последовательный ход: PC<31:2> = PC<31:2> + 1
    - Ветвление: PC<31:2> = PC<31:2> + 1 + SignExt[Imm16]
    - Но при обращении к памяти: Instruction Memory Address = PC<31:2> concat “00”

Combining datapaths

- Каким образом развести различные тракты данных для различных команд?



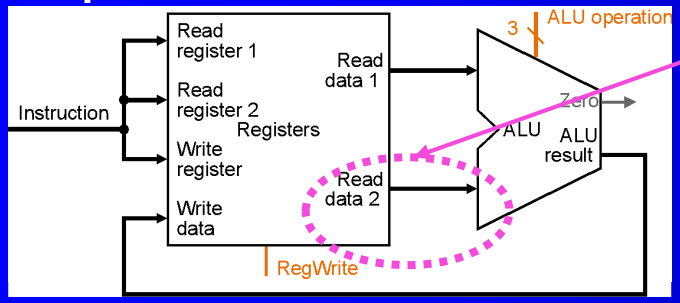
R-type

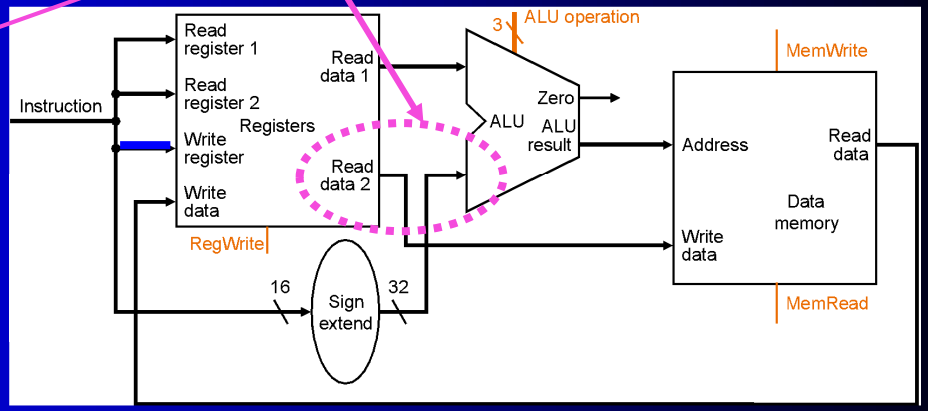


Store

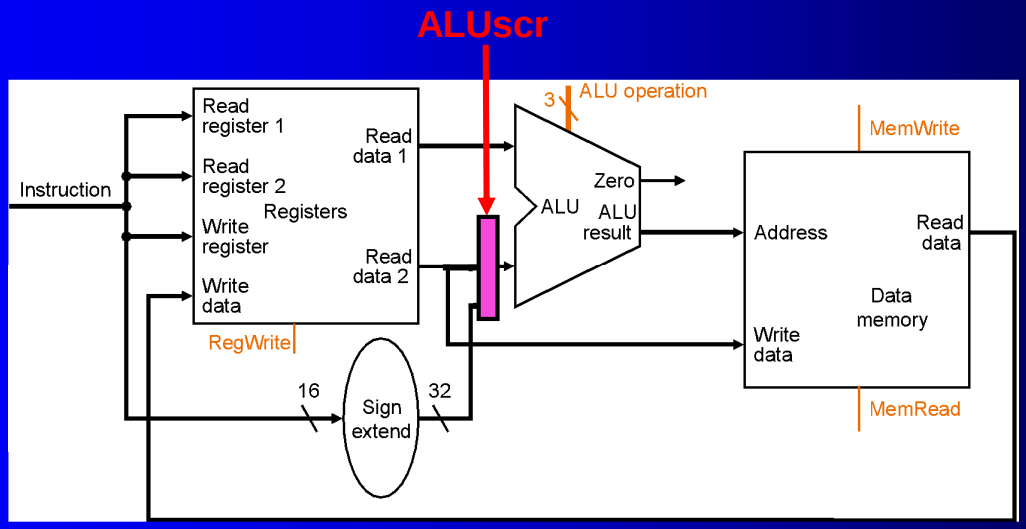
Combining datapaths

Каким образом развести различные тракты данных для различных команд?

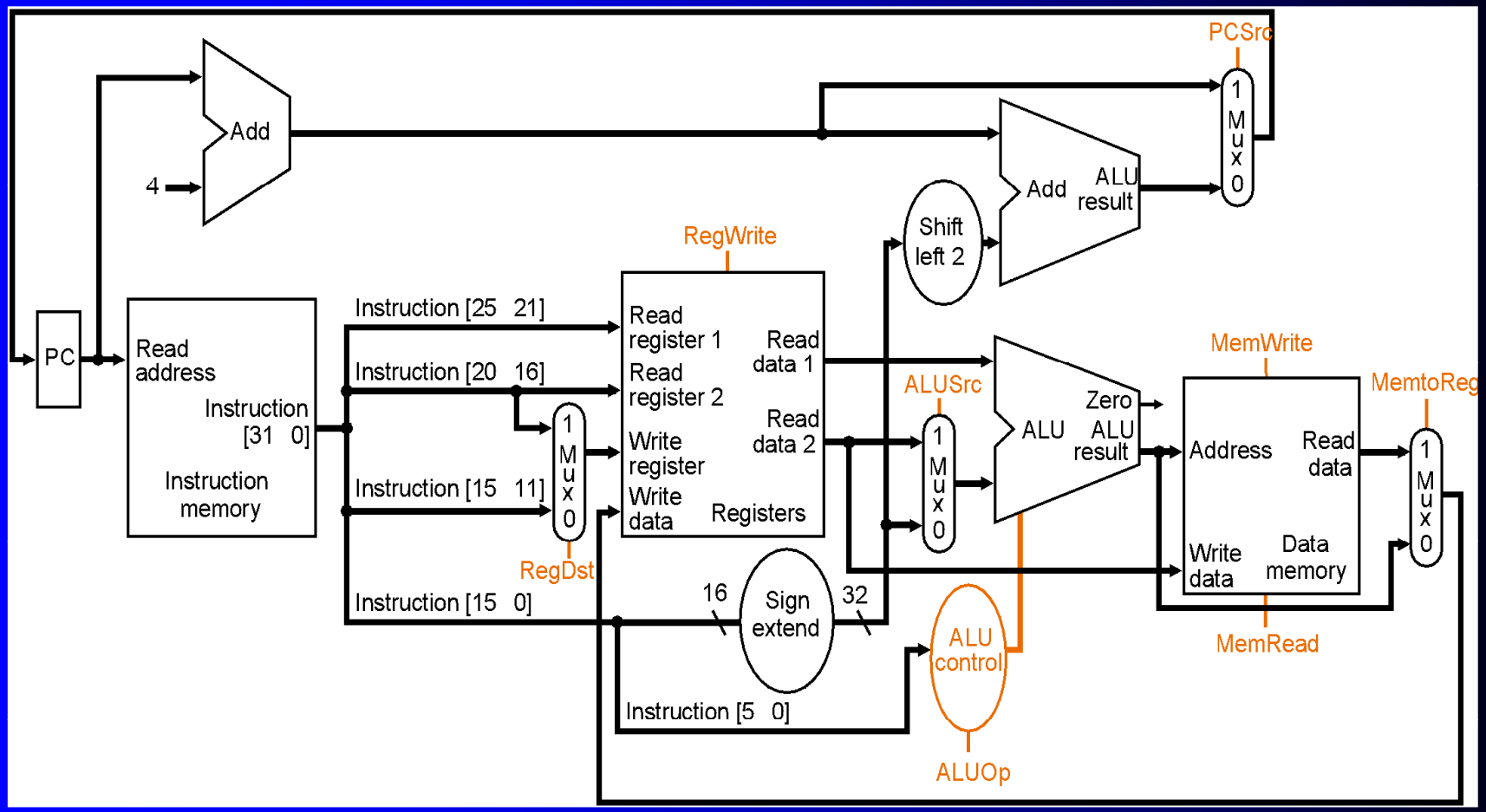




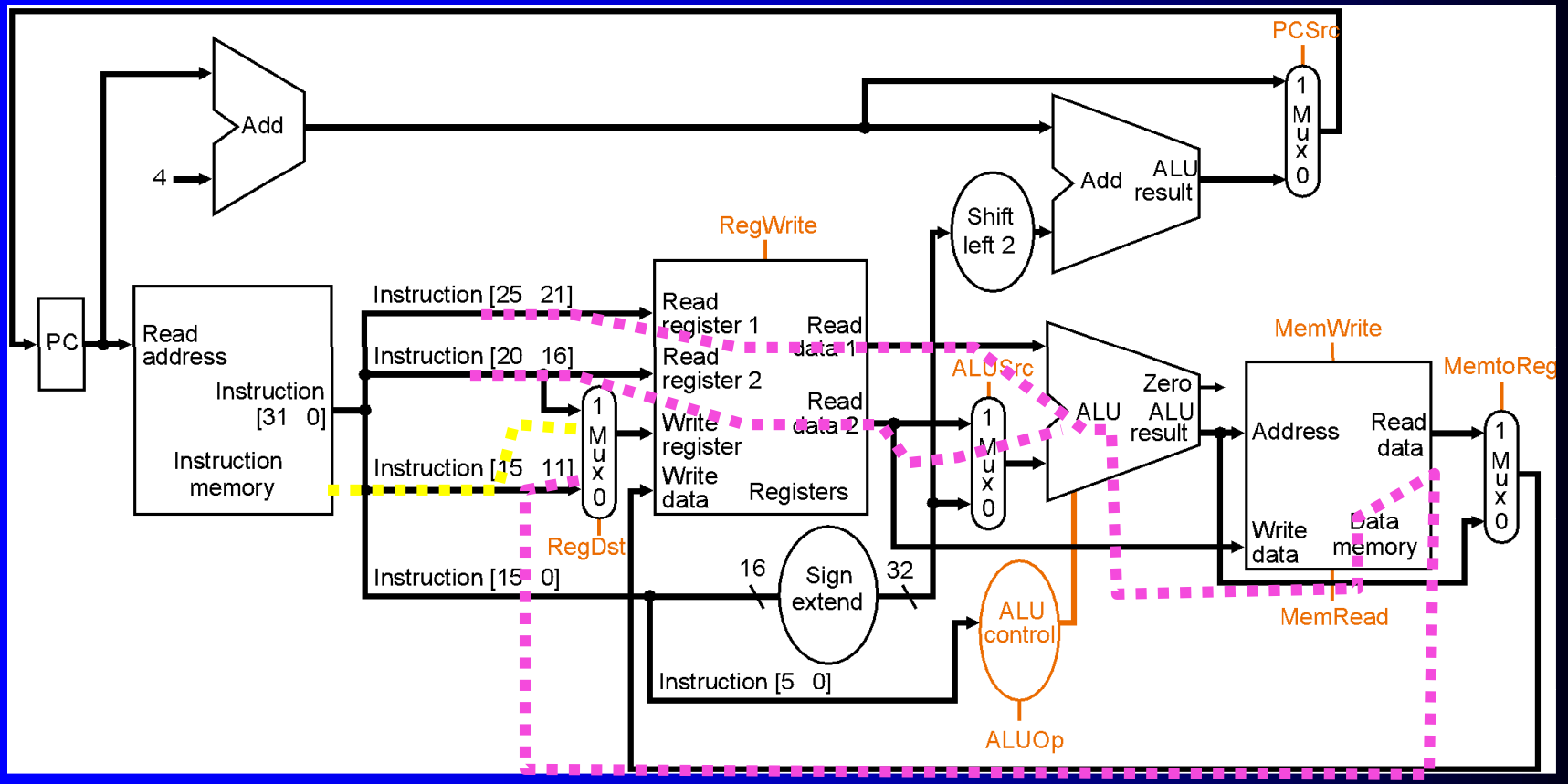
- Используем мультиплексор!



Всё вместе: the single cycle datapath

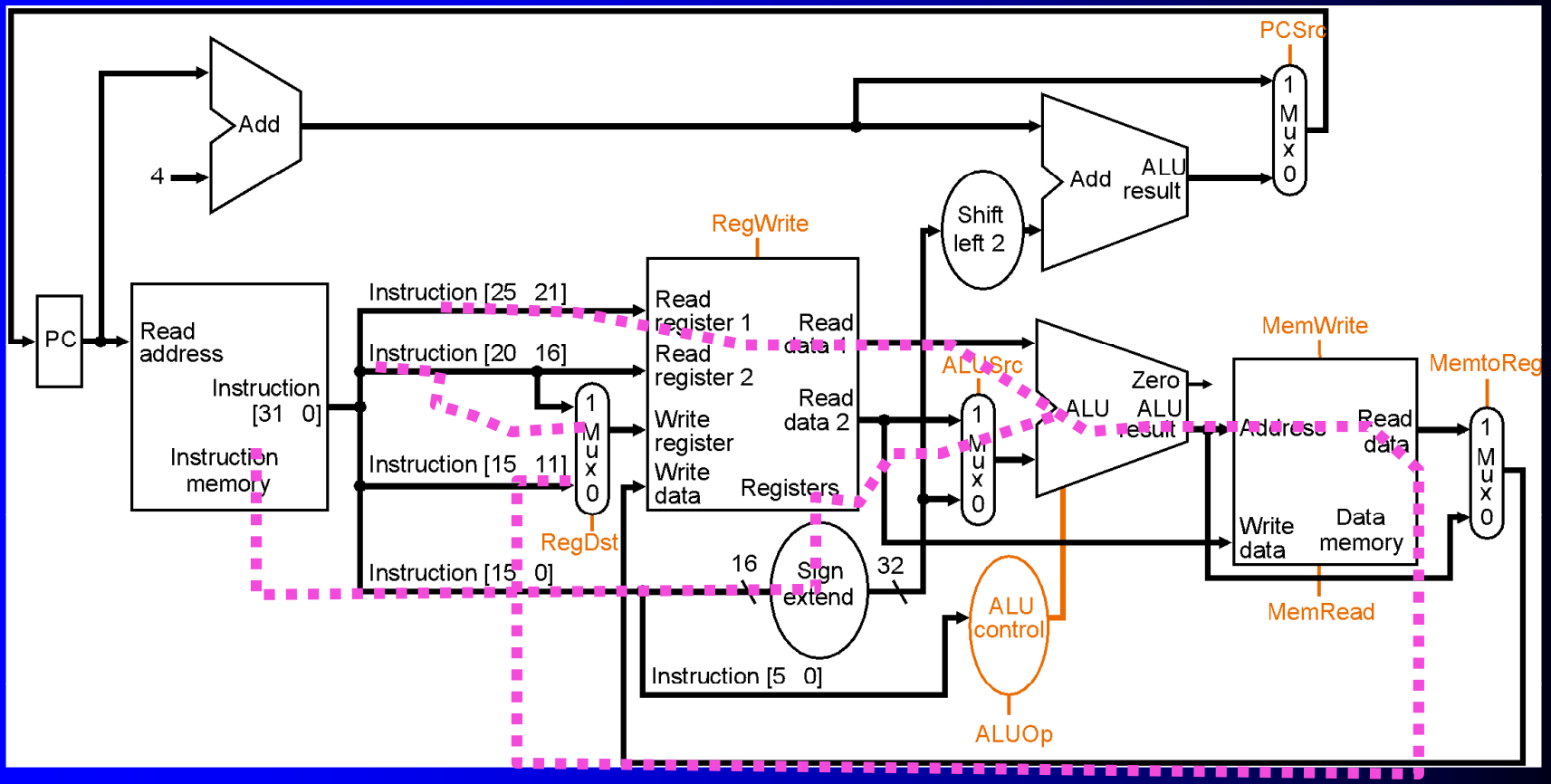


The R-Format (e.g. *add*) Datapath



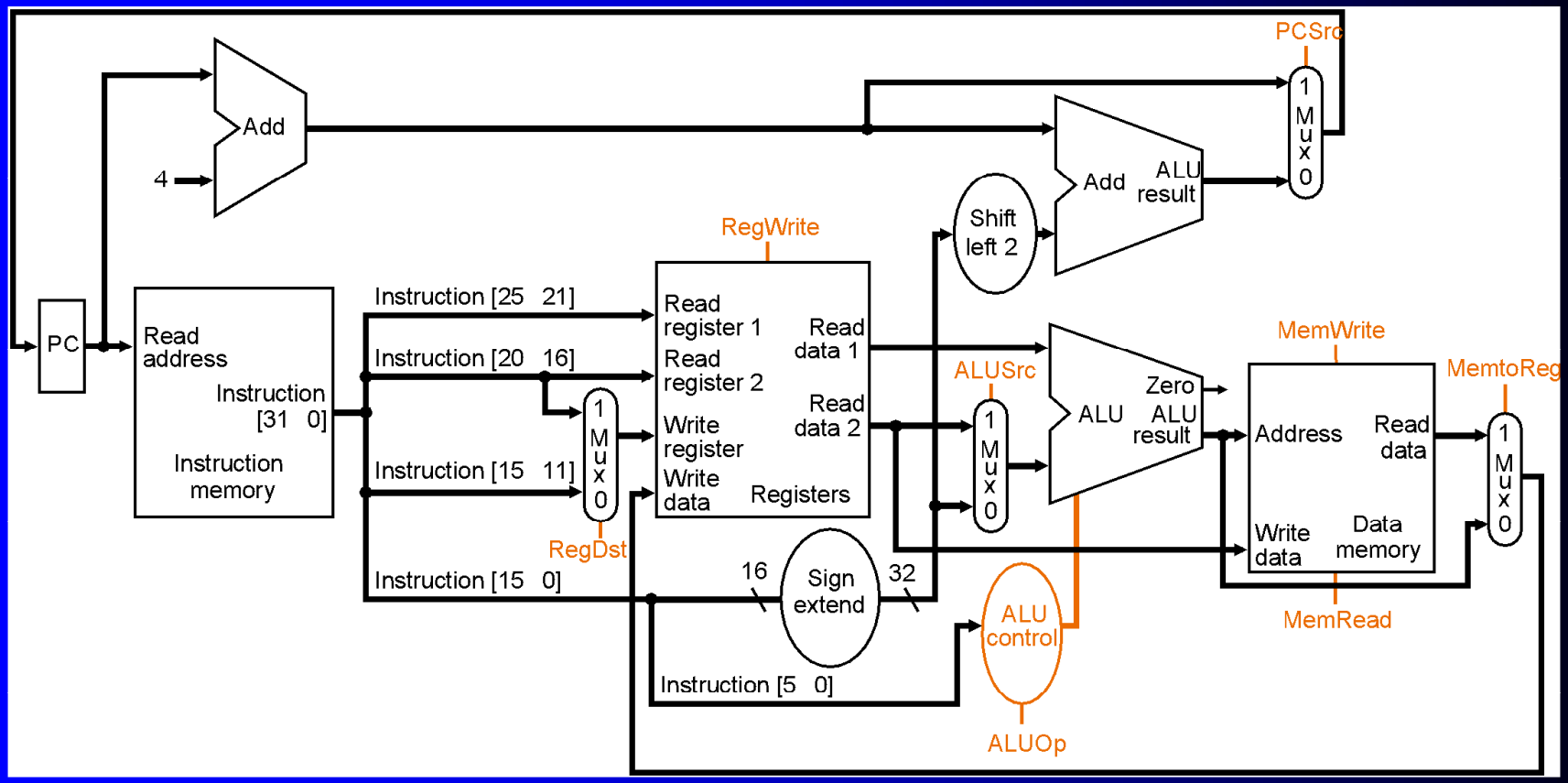
Сигналы УУ ALUsrc=1, ALUop="add", MemWrite=0, MemToReg=0, RegDst = 0, RegWrite=1 and PCsrc=1.

The Load Datapath

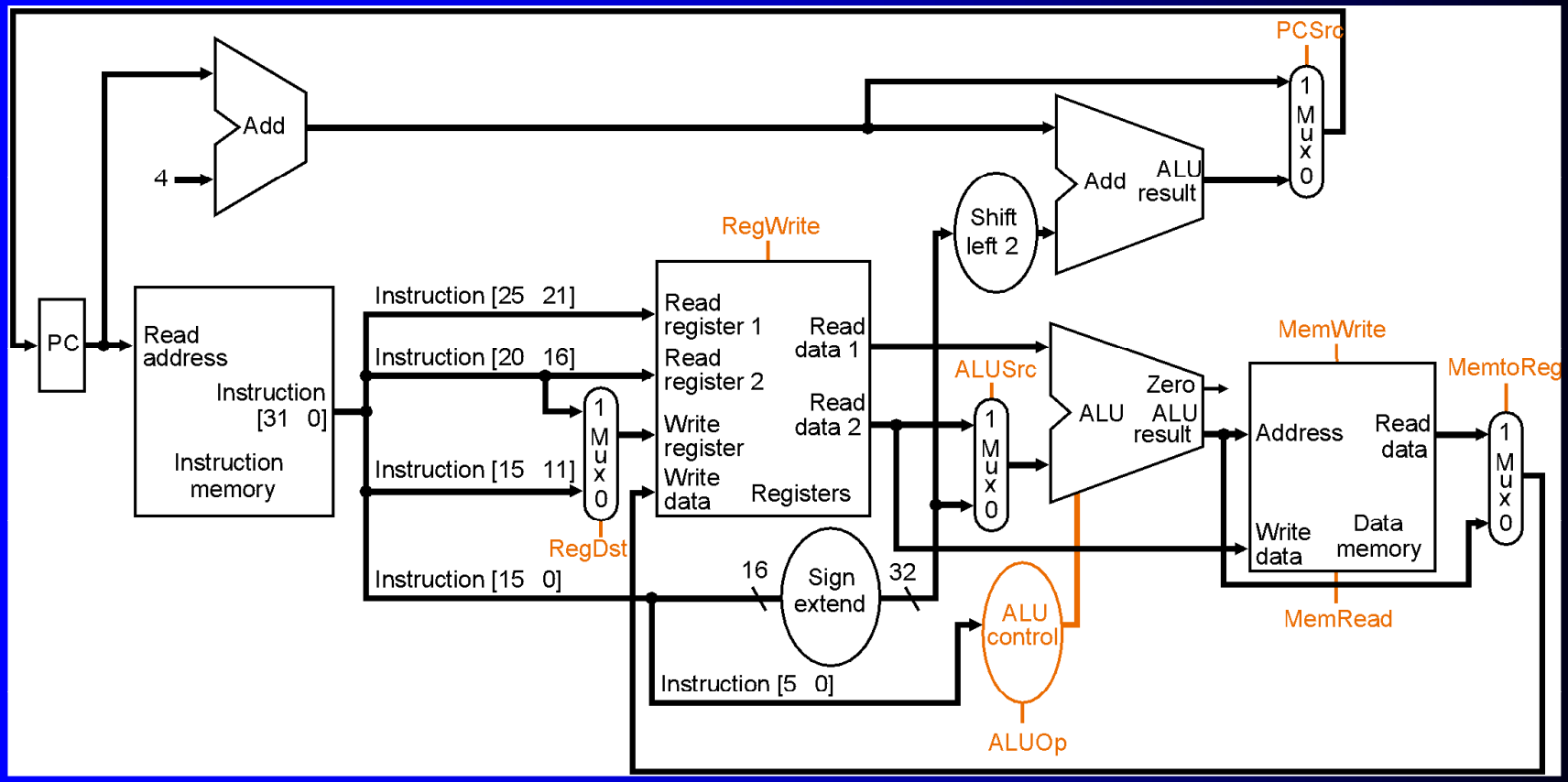


Какие сигналы УУ необходимы для загрузки данных?

The Store Datapath



The beq Datapath



Putting it All Together

- Осталось только сгенерировать управляющие сигналы