Оглавление

Глава 1

Введение

1.1 История

В 1970 г. японская компания хотела сделать калькулятор. Она заказала комплект микропроцессоров компании Intel. Потом другая компания захотела сделать похожий калькулятор. Intel попыталась продать свою разработку и им, но он им также не подошел. Тогда они выпустили свой процессор для всех желающих

1971	4004	8 разрядов	2300 тр., 100 кГц
1972	8008	8 разрядов	3300 тр., С. Джобс, Возник, Apple
1974	8080	8 разрядов	4900 тр.
1978	8086	16 разрядов	29000 тр., 8-16МГц
1986	80386	32 разряда	275000 тр., 32МГц, IA32
1989	80486	32 разряда	1.2млн тр., 100МГц
1995	Pentium	32 разряда	3.1 млн тр.
1997	Pentium III	32 разряда	42 млн. тр
2001	Pentium IV	32 разряда	42 млн. тр, 2ГГц
2007	Core	64 разряда	800 млн. тр

1.2 Структура МПС

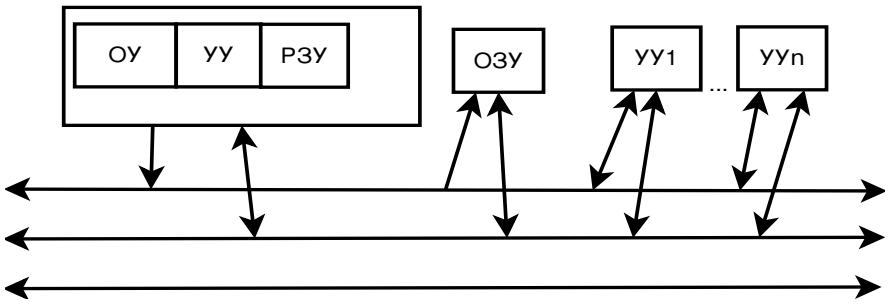


Рис. 1.1: Структура МПС

A – шина адреса (Address)

A	16 → 64кБайта
	32 → 4ГБайта
	64 → 64ГБайта

D – шина данных/команд (Data)

- 8, 16, 32, 64 линии – разрядность обрабатываемых данных

C – шина управления (Control)

- 10-30 линий.

ОУ – операционное устройство
УУ – устройство управления
РЗУ – регистровое ЗУ

ОЗУ – оперативное ЗУ
ИУ – исполнительные устройства
КПР – контроллер прерываний
КПД – контроллер прямого доступа к памяти

Адрес: 1) команд 2) Данных
Управляющие сигналы: — Reset — R/W - чтение запись — I/O - ввод/вывод

1.3 Основные характеристики МП

1.3.1 Основные характеристики МП

1. Разрядность - 8, 16, 32, 64.
2. Объем адресуемой памяти
3. Набор команд.
4. Производительность. Обычно измеряется - MIPS (Million Instruction per Second), максимальная тактовая частота.

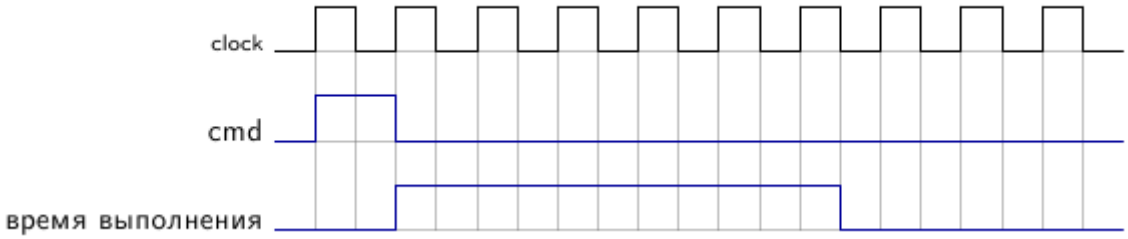


Рис. 1.2: Обработка команд за такт.

Первый сигнал - clock - такты работы микропроцессора.
Второй сигнал - время когда пришла команда.
Третий сигнал - время обработки команды. (6 тактов)

1.3.2 Типы наборов команд

CISC - Complex Instuction Set Computer. 150-400 команд. Pentium, Athlon, Core
RISC - Reduced Instruction Set Computer. 30-150 команд. КПК, PowerPC

1.4 Режимы работы МПС

1.4.1 Выполнение основной программы

Программы и данные в ОЗУ
Процессор:

- Выборка команды
- Выборка данных
- Обработка данных (операция)
- Сохранение результата

1) Адресация очередной команды.
   УУ: адрес → A
   ЗУ: команда → D
   УУ: принять коммадны для исполнения

У процессора есть специальный счетчик PC - Program Counter - **содержит адрес очередной команды.**

$(PC) \to A$.
Выборка команды
$PC + 1 \to PC$ - адрес следующей команды
Команда перехода: новый адрес $\to PC$

---

2) Дешифрация команды.
Формирование последовательности сигналов для её выполнения
Выполнение: 1 или несколько тактов. В каждом такте формируется управляющий код → микрокоманда

3) Адресация операнда.
— РЗУ – формируется номер регистра
— ОЗУ – формируется адрес
УУ формирует адрес в памяти.

4) Выполнение команды.
ОУ: выполнение операции согласно микрокомандам(МК) из УУ.

5) Запись результата.
— РЗУ – формируется номер регистра
— ОЗУ – формируется адрес
УУ формирует адрес в памяти.

Формат

Рис. 1.3: Формат

Группы команд

1. Команды пересылки MOVE
   $R \to R$ (регистр-регистр)
   $R \to M$ (регистр-память)
   $M \to R$ (память-регистр)
2. Арифметические операции
   ADD, SUB, MUL, DIV
3. Логические операции
   AND, OR, XOR, NOT
4. Сдвиг
   Вправо-влево
   Одноразрядные-Многоразрядные
   Лог. сдвиг (заполняем 0)
5. Сравнение

A-B	= 0	: A=B
Знак:	<0	: B>A
	>0	: A>B

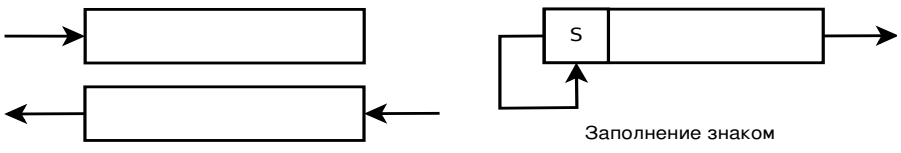


Рис. 1.4: Иллюстрация сдвига

Команды:

1. пересылки
2. арифметические
3. логические
4. сдвиг

Обработки данных. Установка признаков (flags). Регистр состояния StateReg(SR). Для intel'a - EFLAGS
Основные признаки:

1. Z - нулевой результат $z = 1$
2. S - знак. $S = 1(< 0); S = 0(> 0)$
3. V - overflow, переполнение (иногда o). Переполнение при обработке чисел со знаком. $V = C_{n-1} \oplus C_n$.

Рис. 1.5: Регистр состояния lec2_1

1.5 Команды управления

Изменение хода программы
Безусловный переход JMP: $rel \to addr \to PC$ абсолютный ФА = $addr$ (ФА - физический адрес) Относительный переход ФА = $PC + rel$ (смещение)

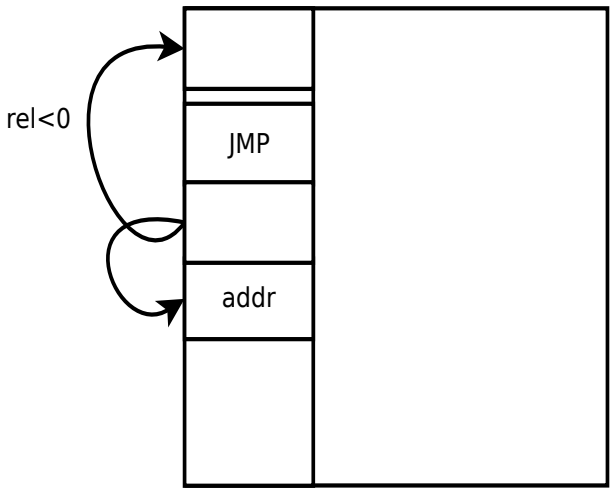


Рис. 1.6: Переход по адресам

1.5.1 Условный переход

по значениям признака
JZ переключается, если Z=1 JZ addr (rel).
JMI addr Mi(минус) S=1: $addr \to PC$
S=0: $PC + 1 \to PC$

1.6 Способы адресации

- безадресные команды.

  NOP, HALT. NOP - не делать ничего, нужна для задержки

- одноадресные.

  Результат сохраняем в той же ячейке. Например $NOT\ addr$

- двухадресные

  Арифметические операции. Результат записывается вместо одного из операндов. Пример:

  $$ADDaddr1, addr2$$

  $(addr1) + (addr2) \to addr1$

- трехадресные

  Более продвинутые двухадресные, позволяющие записать результат в третий операнд. Например:

$$SUB\ addr1, addr2, addr3$$

  $(addr1) + (addr2) \to addr3$

1.6.1 Регистровая адресация

$$ADD\ R1,\ R2$$

$(R_1 + R_2) \to R_1$ Регистров мало, поэтому часто используют адресацию для памяти.

1.6.2 Косвенно-регистровая

$$ADD\ (R1), R$$

$addr = (R1)$

1. Выборка команды.
2. Декодирование
3. Формирование адреса

   Процессор выбрал команды и тут выясняется, что ему нужно взять данные из памяти.
4. Выборка операнда

   $(R1) \to$ ША
5. Выполнение команды операнд $\to$ шина данных $\to$ ОУ (из ЗУ)
6. Запись результата в ОЗУ

Вводится понятие машинного цикла: **Машинный цикл** - период обращения к ЗУ.
Цикл – выборка команды. Любая команда требует как минимум один цикл.

1 цикл – выборка команды

2 цикл – чтение операнда из ЗУ

3 цикл – запись результата в ЗУ

1.6.3 Косвенно-регистровая со смещением

Можем использовать как бы двумерные массивы.

$$NOT\ (R1 + disp)$$

Ф.А. $= (R1) + disp$ – смещение со знаком.

Рис. 1.7: Косвенно-регистровая адресация lec2_3

1.6.4 Прямая (абсолютная) адресация

$$OR\ R1, addr$$

$(R1) + (addr) \to R1$

1.6.5 Непосредственная адресация

Опперанд в команде

$$SUB\ R2, \#5$$

$(R2) - 5 \to R2$.

1.6.6 Относительная адресация (относительно PC)

ФА = $(PC) + rel$ часто в командах. Обычно в командах JMP

1.7 Преимущества и недостатки различных способов адресации. Их использование.

- Регистровая – самая простая и быстрая. Но их мало (в x86 - 8 регистров, в RISC - не более нескольких сотен).
- Прямая – самая долга. Многоуровневая выборка команды.

Рис. 1.8: Переход по адресам lec2_4

- Косвенно-регисровая – мы экономим длину команды, т.к. задаем только номер регистра.

Чаще всего:

1 операнд: регистровая адресация
косвено-регистровая

2 операнд: прямая

Рис. 1.9: Формат команды lec2_5

КОП	2 байта
addr или disp	4 байта
операнд	4 байта

1.8 Вызов подпрограммы

CALL
JSR (Jump to SubRoute).
Отличие CALL от JMP - после JMP не сохраняется адрес возврата.

```
CALL addr. ; PC сохраняется. addr сохраняется в PC  адрес возврата. Или в регистр или в ОЗУ.
...
RET ; Return.
```

Хранение адреса возврата

Рис. 1.10: Хранение адреса возврата lec2_6

Рис. 1.11: Стэк lec2_7

Стэк обычно размещают в конце памяти. Это удобно, потому что после инициализации все - единицы.
Регистр SP (Stack Pointed) - указатель стека. Этот регистр хранит адрес **вершины** стэка. Это важнейший служебный регистр.

CALL: $P \to (SP)$
$SP - 1 \to SP$
CALL: ...
RET: $SP + 1 \to rSP$
$(SP) \to PC$

1.9 Обслуживание прерываний

Процессор прерывани выполнение и переходит к **специальной программе – обработчику прерываний** (Interrupt Handler).

Причины:

1. аварийные ситуации. (немаскируемые).
2. штатные (маскируемые). Какое-то устройство что-то попросило (передача и прием данных).

1.9.1 Программные команды

Команды: INT (Interrupt в процессорах Intel)
SWI (Software Interrupt)

- Сохранение контекста (PC, SR, адресных регистров)
- вызов специальной программы обработки (фиксированый адрес).

1.9.2 Аппаратные

- запрос внешних устройств - IRQ (Interrupt request)
- ошибки функционирования: деление на 0, неправильный код команды, неправильное обращение к памяти, нарушение привелегий пользователей. Программа пытается обратится не туда и генерируется исключение (Exception)

Рис. 1.12: Прерывания lec2_8

1.9.3 Приоритеты обслуживания

Рис. 1.13: Стэк lec2_9

– Фиксированой приоритет.

IRQ0 - максимальный. IRQn - минимальный.

Не всегда удобно, т.к. при переходе от программы к программе, приоритеты могут меняться.

– Программируемый приоритет.

Управляющие слово → КПР

В таком случаи КПР обслуживается как устройство. Процессор посылает командой move соответствующие команды в КПР.

– циклический приоритет.

Если устройство обслужили, оно получает минимальный уровень автоматически, становится в конец очереди. так обеспечивается равномерность.

1.9.4 Процедура обслуживания

– Приход IRQ_i.

– КПР → сигнал INT → МП.

Процессору поступает сигнал INT. МП должен проверить разрешены ли запросы или они маскированы.

– Завершает выполнение команды.

– Сигнал подтверждения.

Вызывает сигнал подтверждения для КПР - INTA (Acknowledge).

– Сохранение контекста.
  $PC, SR \rightarrow$ стек

– загрузка в PC **вектор прерываний** – адрес первой команды обработки прерываний.
  Специально делают таблицу векторов для каждого источника.

– Запрос в ОЗУ за таблицей векторов прерываний.

Каждый IRQ → вектор
запрос

КПР - отдельная микросхема. Если источников немного - встраивается в процессор.

1.10 Прямой доступ к памяти

DMA - Direct Memory Access. Задача: обеспечить прямой обмен между ОЗУ и внешними устройствами.

1.10.1 Программный обмен

1. ввод ИУ → МП
2. запись МП → ОЗУ.

1.10.2 Прямой доступ к памяти

1. Отключение МП
2. Обмен ВУ $\Longleftrightarrow$ ОЗУ

Отключение МП: A, D переходят в **третье состояние**.

1.10.3 Обслуживание DMR

1. Поступил DMR.
2. КПД → сигнал HOLD к МП. HOLD - захват шины
3. МП – завершает цикл и отключается от шины адреса и посылает HOLDA.
4. КПД принимает управление системой.
   – Формирует адрес ячейки ОЗУ
   – формирует управляющие сигналы. R/W для ОЗУ и I/O для ИУ.

Чтобы КПД мог выполнять свои функции, он предварительно программируется.

Предварительное программирование КПД

– запись начальных адресов ОЗУ

– указать объем массива передаваемых или принимаемых данных.

– режим приоритета запросов

КПД – автоматически формирует адрес следующей ячейки ОЗУ. По окончании массива:

- снимает HOLD
- МП прдолжает работу

Глава 2

Функционирование устройств

2.1 Операционное устройство

арифметичские + логические (АЛУ.) + сдвиговые (сдвигатель)

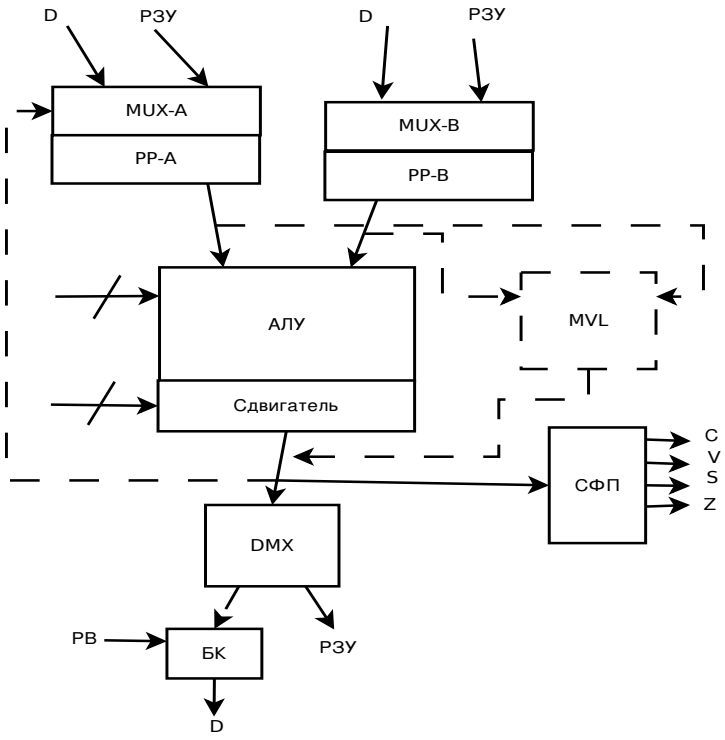


Рис. 2.1: ОУ

2.1.1 Основные арифметические операции

- +, -
- INC(+1)
- DEC(-1)
- NEG - изменение знака
- Перевод в доп. код
- x= $\overline{A} + 1$
- MUL, DIV – последовательный +, - и сдвиг.
- MVL – блок ускореного умножения
- Логические операции AND, OR, XOR, NOT.

2.1.2 Аккумуляторные структуры

РР-А - регистр-аккумулятор. А - один из операндов или результата Когда не происходит передача данных на шину D от неё необходимо отключиться. Поэтому подключение идет **Обязательно подключение через буферный каскад с треями состояниями!**. 0, 1 - данные, отключение от шины.

РВ - сигнал разрышения вывода.

2.1.3 Управляющий сигнал

Микромкоманда ОУ → УУ. 10-20 разрядов.

2.2 Устройство управления. Control unit

2.2.1 Что необходимо сделать?

- Выборка команды. Адрес → из РС.
- Дешифрация последовательности МК для выполнения.
- Формирование адреса операнда.
- Запись результата.
- доп. функции (формирование прерываний и так далее).

2.2.2 Блоки

УУ делится на Блок формирования адреса (БФА) и блок формирование микрокоманд (БФМК).

Принцип работы БФА

Источники адреса:

- адрес команды в РС
- адрес операнда → зависит от способа адресации:

Тип	источник
прямая	шина D
косвеный регистр	РЗУ
Косвеный регистр со смещением	Адресный сумматор (РЗУ + $disp$ → адрес)
Относительная	$(PC) + disp$ → адрес
Адресация стека	SP

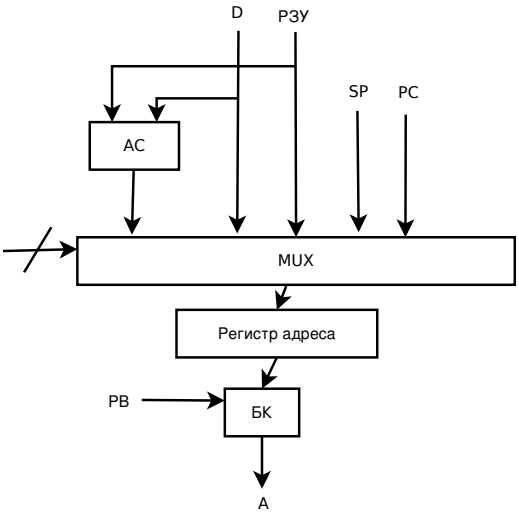


Рис. 2.2: БФА

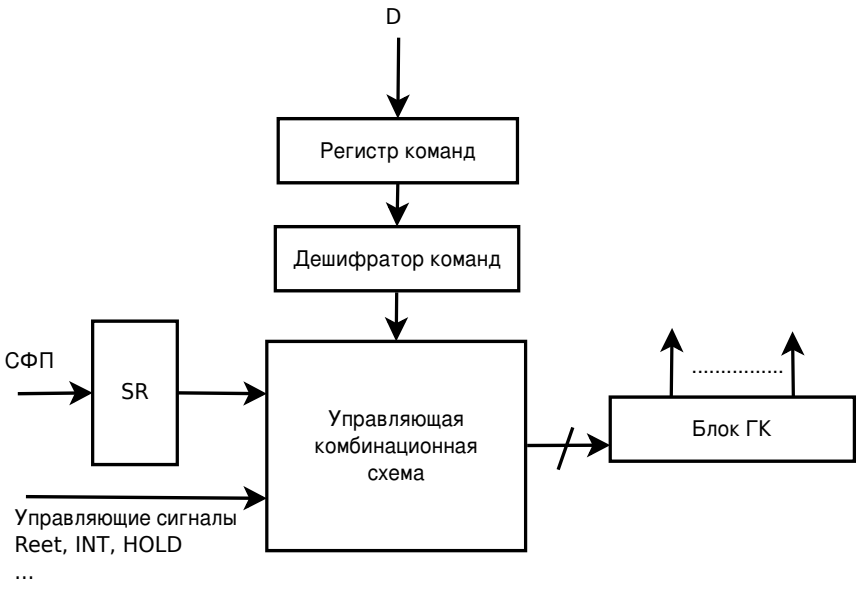


Рис. 2.3: БФМК

По идее для каждой команды должен быть свой ГК, но это слишком дорого → до 100-150 разрядов.
Длина последовательности от 1 до 30 мк.

1. БФМК бывают на базе ГК:

   Для сокращения сложности используют управляемый генератор кодов (УГК). УГК – формирует несколько последовательностей. Испольуется при простых наборах команд.

2. на базе микропрограммируемого ЗУ (МПЗУ).

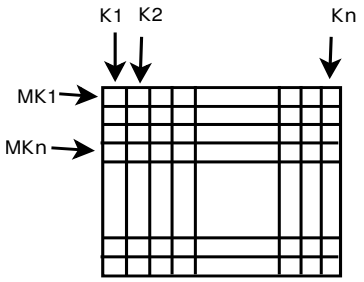


Рис. 2.4: МПЗУ

Число команд ~ 200, длина ~ 5, разрядность МК ~ 100 → МПУ $\sim$ 100кбит, 16кбит, но память должна быть **очень** быстрая.

Проблема: минимизация МПЗУ – чем меньше, тем быстрее булет работать процессор.

Микропрограммирование – эффективная кодировка, эффективное размещение.

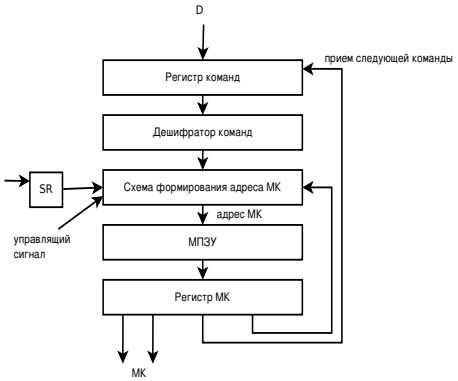


Рис. 2.5: Х.з.

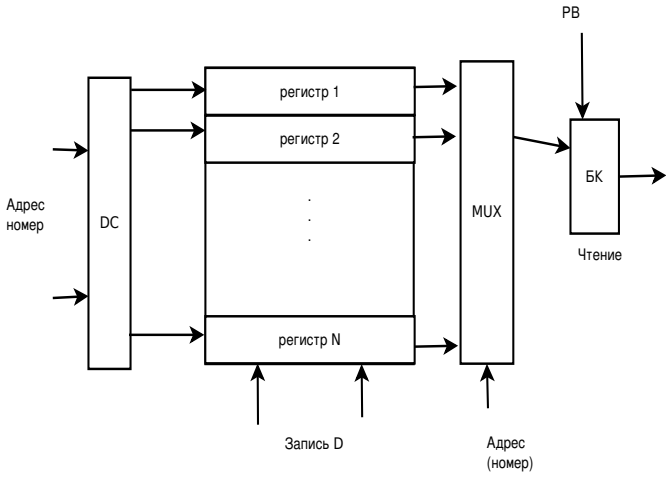


Рис. 2.6: Х.з.

2.3 Оперативные ЗУ

Нам нужно 256МБайт, а микросхемы 64МБит x 1. Чтобы выбирать байт нам нужно объединить сразу 8 микросхем.

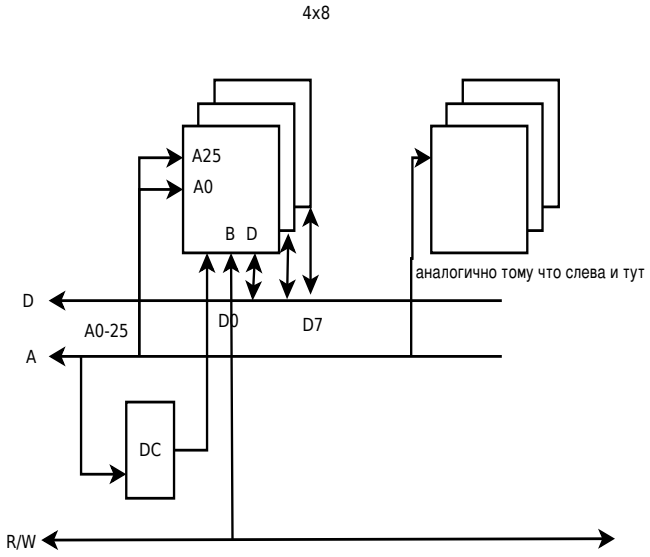


Рис. 2.7: Схема работы ОЗУ

Сигнал R/W генерирует УУ.

Глава 3

Интерфейсное устройство

Порт - простейшее устройство параллельного ввода/вывода

3.1 Порт ввода данных

Утройство – регистр и буферный канал

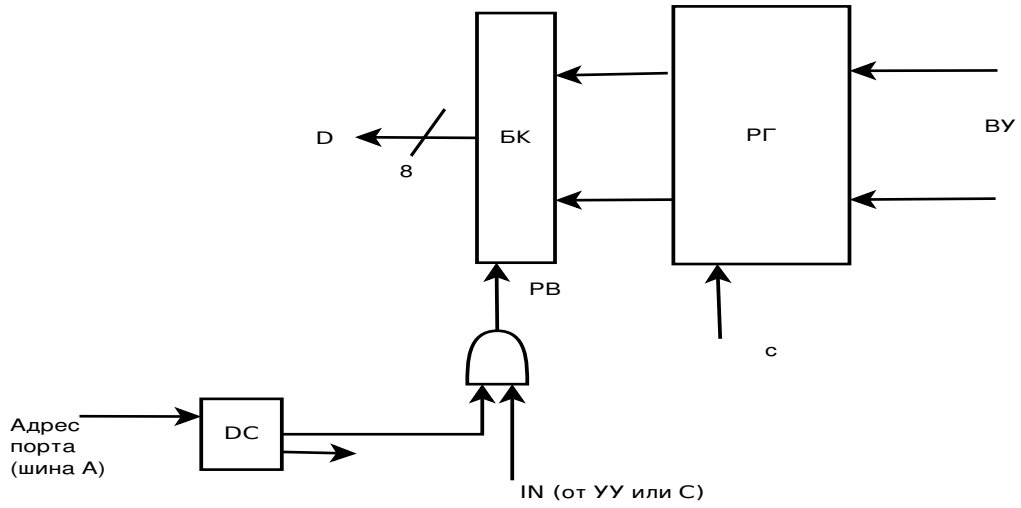


Рис. 3.1: Входной порт

Устройство – регистр и буферный каскад. Разрешение о выдаче формируется в зависимости от адреса порта. Сигнал ввода данных, который дается по шине управления сигнала от УУ. То есть шина С имеет отдельные сигналы для управления портами.

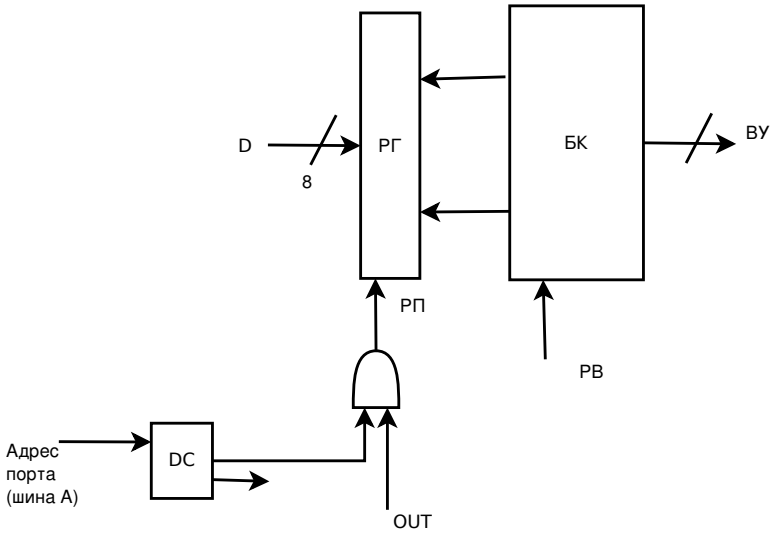


Рис. 3.2: Выходной порт

Адрес порта задается по шине А.

3.2 Двунаправленные порты

Часто используются двунаправленые порты, которые позволяют...

Причем часто используются возможности индивидуального программирования вывода, то есть мы можем сделать половину выводов входами, а половину выходами. Управляющее слово имеет столько же разярдов, сколько соответсвующи порт выводов. Управляющие слово: $C_7...C_0$, $C_i = 0-$ **ввод**, $C_i = 1 -$ вывод.

Мы должны сперва загрузить управляющее слово в соотвествующий регистр и потом уже использовать выводы как входы или выходы

Глава 4

Классификация микропроцессоров

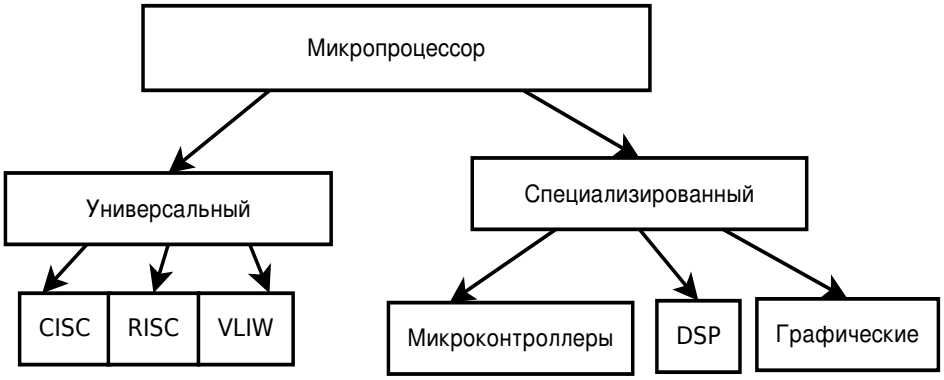


Рис. 4.1: Классификация

По объемам выпуска (2008 год): Универсальные процессоры ∼500млн. в год. (∼40млрд $). Микроконтроллеры: ∼4млрд. в год (∼15млрд $) DSP: 500млн. (∼ 15млрд.$)

Глава 5

Микроконтроллеры

Специальные Микропроцессоры для выполнения функций управления объектами.

5.1 Большая номенклатура

Интегрированные на кристаллы памяти и переферийных устройств
Однокристальные микроЭВМ
Патент был получен в 1981 г. Кочрен, Бун.
Крупные производители:

1. Renesas (Hitachi, Mitsubisi) 19%
2. Freescale Semiconductor (быв. Motorola) 16%

Микроконтроллеры бывают 8, 16 и 32 разрядными. Относительно простые задачи - 8 разядные.
CISC:
MCS-51 (Intel, Atmel,...), семейство 8051. HC-6808 (Freescale)
RISC:
AVR(Atmel), PIC(Microchip).
Для решение достаточно сложных задач управления используются 32-х разрядные микроконтроллеры.
Наиболее распространены: AMR(Atmel, NXP, Samsung, TI, ...)
PowerPC(Freescale)
MIPS(Freescale)
16-и разрядные не очень популярны, но если используются, в основном TI MSP430.

5.2 Общая структура МК

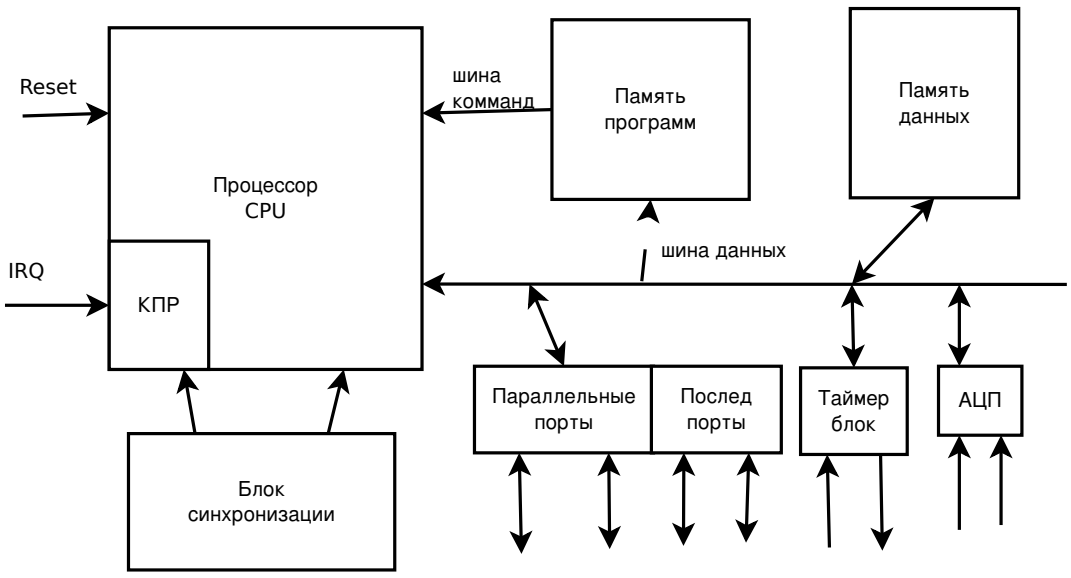


Рис. 5.1: Общая структура МК

- CPU – Central Processing nit
- Память команд – память данных и команд в Гарвардской архитектуре раздельная. Бывает ПЗУ (ROM), программируемая при изготовлении. Или FLASH-память.
- Параллельные порты – 8-и разрядные. От 1 до 10.
- Последовательные порты – внешняя информация поступает и выдается последовательно. Часто используются SCI и SPI.
- Контроллеры стандартных шин – часто USB, Ethernet CAN.
- Таймерные блоки – 1-3 таймера, таймерные процессоры (содержат более 16 таймеров).
- АЦП – 8-12 разрядные.
- Блок синхронизации – генератор тактовых импульсов. Обычно подключается внешний кварцевый генератор, а блок синхронизации умножает или делит частоту.
- Специальные блоки – криптографические блоки, коммуниационные процессоры (реализуют разные протоколы обмена).

5.3 Семейства МК

Обычно это 10-40 моделей. Общий CPU. Отличаются набором других блоков:

- Объем памяти. От Кбайт до Мбайт.
- Различное число портов (выводов).
- Различные колличества АЦП.
- Различные дополнительные контроллеры.

5.4 Стоимость МК

5.4.1 8 разрядов

1-10$ (<1$). Если есть какие-то контроллеры, например USB, может возрастать до 20-25$

5.4.2 32 разрядные

10-40$. Очень сложные могут стоить до 100$.

5.5 Различные типичные характеристики МК

FLASH от 1КиБ до 1МиБ
ОЗУ данных от 100 байт до 100КиБ

Глава 6

Микроконтроллеры семейства 68HC08

6.1 Процессор CPU08

Аккумуляторная архитектура

6.1.1 Регистровая модель

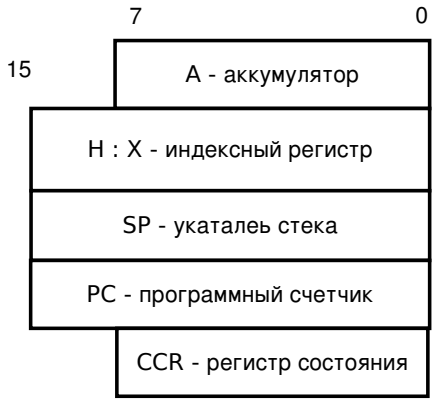


Рис. 6.1: Регистровая модель

- ОЗУ до 64Кбайт, память данных – до 64Кбайт.
- Индексный регистр – адрес операнда в ОЗУ.
- SP содержит указатель на вершину стека
- CCR - Condition Code Register – по сути регистр состояний.

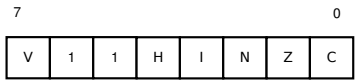


Рис. 6.2: Структура CCR

- V - переполнение.
- H - перенос между тетрадами (HalfCarry)
- I - маска перерывания (0 - обработка IRC, 1 - не будут)
- N - знака
- C - перенос.

6.1.2 Способы адресации

1. Регистровая – Для хранения данных можно использовать только регистр A, поэтому способы адресации ограничены. Младший байт – X отдельно выделен на картинке. Обычно он выдает младший байт адреса, но можно иногда использовать его для хранения операндов.
2. Косвенно-регистровая (индексная) – адрес $H:X$.
3. Индексная с постинкриментов – $(H:X)+1 \to H:X$.
4. Индексая со смещением – адрес $= (H:X) + d_8, d_{16}$; d8 или d16 - смещение со знаком.
5. Индексая со смещением с пост-инкриментом – адрес $= (H:X) + d_8, d_{16} + 1$.
6. прямая – адрес в команде
7. непосредственная – операнд в команде
8. индексация по SP со смещеним: адрес $= (SP) + d_{8,16}$

КоП, 1,2 или 3 байта.

Пример

INC(опер): (опер) + 1 → опер
INC A, INC X - увеличение на 1 регистров.
Для косвенной INC X
Для индексной с пост-инкриментом: INC X+
Если хотим выбрать X со смещением: INC d,X
Из индексной со смещением и с постинткриментом: INC d,X+
Прямая адресация: INC addr
Признак 16-и ричного числа является символ \$: \$10 = 16_{10}
Если мы задаем не адрес, а операнд, ставится признак #.
Если индексация по SP, пишется: INC d,SP
Пример: $ADD\#10$

6.1.3 Команды

Команды пересылки

Значение	Команда	Пример	комментарий
Загрузка (LOAD)	LDA LDX LDHX	LDA (опер) LDX (опер) LHDX (опер)	Загрузить в A Загрузить в X $M \to H:X$
Сохранение (STORE)	STA STX STHX MOV	STA (опер) STX (опер) STHX (опер) MOV addr1,addr2	$A \to M$ $X \to M$ $X \to H:M$ память → память
регистр-регистр	TXA TAX TAP TPA TSX TXS		$x \to A$ $A \to X$ $A \to CCR$ $CCR \to A$ $SP+1 \to H:X$ $H:X-1 \to SP$
Очистки	CLRA CLRX CLRHX CLR	 CLR (арг)	$0 \to A$ $0 \to X$ $0 \to H:X$ 0 → арг – очистка ОЗУ
Загрузка в стек	PUSHA PUSHX PUSHH		$A \to (SP), SP-1 \to SP$ $X \to (SP), SP-1 \to SP$ $H \to (SP), SP-1 \to SP$
Извлечение из стека	POPA POPX POPH		$(SP+1); SP \to A$ $(SP+1); SP \to X$ $(SP+1); SP \to H$

Арифметические операции

Сложение	ADD (apr) ADC (apr)		$A + M \to A$ $A + M + C \to A$
Вычитание	SUB (apr) SBB (apr)		$A - M \to A$ $A - M - C \to A$
Умножение	MUL (apr)		$A * M \to X : A$
Деление	DIV (apr)		$H : A \div M - > A$, ост$- > H$
Инкримент	INCA INCX INC (apr)		
Декремент	DECA DECX DEC (apr)		A - 1 -> A X-1 -> X M-1 -> M
Изменение знака	NEGA NEGX NEG		0-A -> A 0-X -> X 0-M -> M
	AIX AIS	AIX #Im AIS #Im	H:X + Im -> H:X SP + Im -> SP
Сравнение	CMP (apr) CPX (apr) CPHX (apr)		A-M установка признака X-M H:X-M
Тестирование	TSTA TSTX TST (apr)		A-0, Установка N,Z X-0, Установка N,Z M-0, Установка N,Z

Логические операции

Логические (Поитовые)	AND (apr) OR (apr) XOR (apr) COMA COMX COM (apr)	A+M -> A	A*M -> A $A \oplus M$ -> A $\overline{A} - > A$ $\overline{X} - > X$ $\overline{M} - > M$
сдвги	LSLA LSLX LSL (apr) LSRA LSRX LSR (apr) ASRA ASRX ASR (apr) ROLA ROLX ROL (apr) RORA RORX ROR (apr)		логический влево логический влево логический влево логический вправо логический вправо логический вправо арифметический вправо арифметический вправо арифметический вправо циклический влево циклический влево циклический влево циклический вправо циклический вправо циклический вправо
битовые операции	CLRC SEC CLI SEI BCLR n,(apr) BSET n,(apr)		0->C 1->C 0->I 1->I 0 -> bn 1 -> bn

Команды управления

Переход BRA rel8	JMP (apr)	$PC+rel \to PC$, rel - смещение со знаком	$M \to PC$
Условные ветвления	Bcc rel		Если CC, $PC + rel8 \to PC$ CC смотри ниже

CC	Условие	Пример	комментарий
NE	Не равно, z=0	BNE	
EQ	Равно, z = 1	NEQ	
PL	>0, N=0	BPL	
MI	<0, N=1	BMI	
HI	> Z+C = 0		без знака
LO	< C= 1		без знака
LS	≤ Z+C = 1		без знака
HS	≥ C = 1		без знака
BGT	$> Z + (N \oplus V) = 0$		знаковое
BLE	$<= Z + (N \oplus V) = 1$		знаковое
BGE	$>= N \oplus V) = 0$		знаковое
BLT	$< N \oplus V) = 1$		знаковое
HCC	H = 0		
HCS	H = 1		
MC	I = 0		
MS	I = 1		
IH	сигнал IRQ = 1		
IL	сигнал IRQ = 0		

Ветвление по значению бита

CC	Условие	Пример	комментарий
BRCLR	если бит n = 0	BRCLR n,(opr),rel8	
BRSET	если бит n = 1	BRSET n,(opr),rel8	
CBEQ	Если A = M	CBEQA (opr),rel8	
CBEQA	Если A = #Im	CBEQA #Im,rel8	
CBEQX	Если X = #Im	CBEQX #Im,rel8	

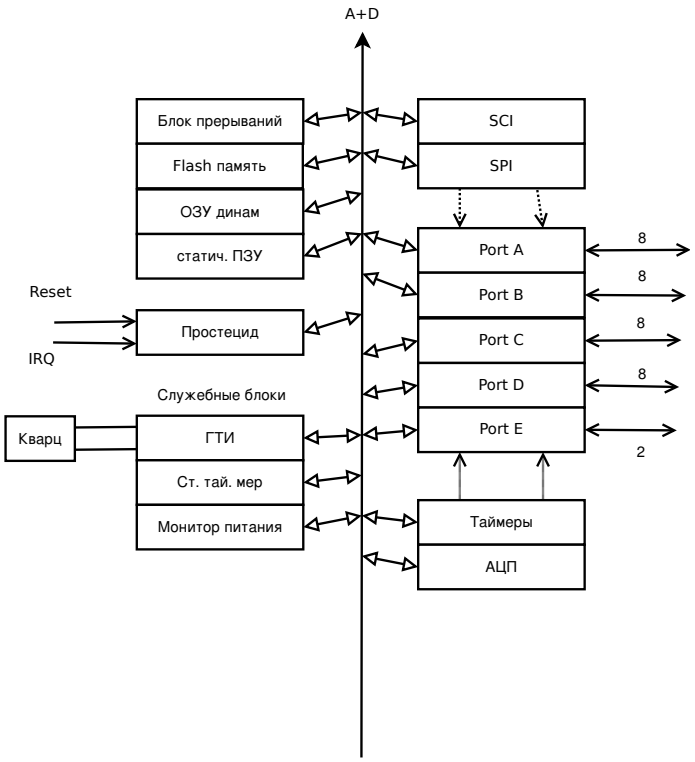
Организация циклов

CC	Условие	Пример	комментарий
DBNZ	$M - 1 \to M; PC + rel8 \to PC$, если z=1	DBNX (opr),rel8	

6.1.4 Энергосберегающий режим

Рабочий режим 15мА
Ожидание: 4мА
Остановка: 3мкА, но запуск будет через

6.1.5 Стркутура МС



Flash – хранение программ
ОЗУ – данные
ПЗУ – служебная память (загрузчик, отладчик)
Ст. таймер – сторожевой таймер (watchdog).
Монитор питания – прерывание, при понижении питания ниже заданного уровня
ГТИ - генератор тактовых импульсов
Последовательные порты – последовательные порты.
Параллельные порты - параллельные порты :)
Таймеры и АЦП – имеют выходы через параллельные порты.

6.2 Функционирование процессора при выполнении команд

6.2.1 Выборка команд

РС -> ША, команда из flash в Шину Данных БФА. УУ принимает и декодирует. Если команда 2 или 3 байтная, нужно получить эти байты.

РС+1 -> РС -> ШД – 2 байта -> ШД принимает (нужно написать кто принимает и зачем он нужен) РС+1 -> РС -> ШД – 3 байта -> ШД принимает

Команда принята, нужно её выполнять. Выполнение -> АЛУ.

АЛУ делает что нужно – складывает, делает логические операции, сдвиги и так далее.

Если нужно выбрать операнд из ОЗУ, то нужно предварительно: БФА -> ША.

БФА сформировал адрес с учетом смещения, ОЗУ выдает данные: ОЗУ -> ШД.

Этот операнд идет в АЛУ для обработки. (эти две операции - цикл)

Если после выполения результат нужно записать в память:

БФА -> ША. АЛУ -> ШД. ШД -> ОЗУ.

Выборка команды: 1,2, 5 циклов в зависимости от длины команд.

Чтение операнда из ОЗУ – +1 цикл чтения.

Запись результата в ОЗУ – +1 цикл чтения.

ASR -10,X КОП: -10 – 2 байта.

Выборка команды - 2 цикла.

1. РС -> ш. А, выборка коп РС+1 -> ш. А, выборка -10
2. БФ формирует адреса (H:X)-10 -> ш. А, операнд -> ш. D. Выполняется сдвиг.

3. Запись результата БФА: (H:X)-10 -> ША, АЛУ -> ШД, запись ОЗУ.

   Изменения в CCR: N - не меняется
   C - младший бат оператора

   ADD 1000: Байтов: 3 байта (КОП + 2 байта на 1000) Циклов: 1) выборка 2 цикла (PC -> ш. А - выборка КОП, PC+1 -> ш. А - выборка первой части 1000, PC+1 -> ш. А - выборка второй части 1000) 2) Данные -> АЛУ. Запись в А.

   Регистры: PC+3 SP -> SP H,X -> H,X A -> A + $1000 CCP: V, H, Z, N, C могут изменится.

   PUSHA: Байтов: 1 Циклов: 2

   1) Запись А в адресс $SP: SP -> ША. А -> ШД. ШД -> ОЗУ 2) SP-1 -> SP

6.3 Параллельные порты

Основные средства связи с объектами управления (до 10 портов).
Двунаправленые порты: ввод информации от объекта (состояние системы). Порты могут работать на ввод и на вывод данных ваш Кэп. На выводе передаются управляющие коды, которые меняют состояние каких-то обхектов, подключеных к МК.

- Port A – 8 i/o
- Port B – 8 i/o -> входы АЦП
- Port C – 7 i/o
- Port D – 8 i/o -> Таймеры, посл. порт SPI
- Port E – 2 i/o -> посл. порт SCI

Рис. 6.3: Структура адресса

6.3.1 Фиксированые адреса

PTA $0000 -> DDRA – $0004 PTB $0001 -> DDRB – $0005 PTC $0001 -> DDRC – $0006 PTD $0003 -> DDRD – $0007 PTE $0008 -> DDRE – $000C

1. Установка режима – запись в DDRx. На каждый вывод можно установить запись-чтение.

   Если 0 - ввод, если 1 - вывод. Устанавливается число 8 бит (нумерация с 0 до 7).

2. по Reset 0 -> DDRx (ввод).

Команда STA addr – надо указать адрес порта.

```
 lda #Im ; код режима
sta addr
```

6.3.2 Пересылка данных

Ввод: lda addr_port port -> A
Сохранение: sta addr port x -> ОЗУ

6.3.3 Вывод данных в порт

sta addr_port a->port x

6.3.4 Особенность

«Подтягивание» выводов к напряжению питания

Рис. 6.4: Схема

Ставится транзистор, который в случаи сигнала подтягивает его к питанию.
Регистр: PTxPUE – Pull-Up: x=1 – подтягивание. x=0 – не подтягивание
Плюсы данного подхода:

1. Исключить «свободные» выводы, чтобы они не ловили помехи.
2. Объединение выводов. «Монтажное И».

6.4 Последовательные порты (адаптеры)

Преобразование параллельного представления данных в последовательное и обратно:

6.4.1 Развитие сетевых технологий

SCI – Serial Communication Interface – последовательный связной интерфейс.
Данный интерфейс широко применяется.
SCI – упрощенный вариант UART (Universal Asynchronus Recive-Transmitter).
UART – стандарт RS-232 (поддерживает модемную связь - телефонные линии)
SCI работает также как UART. Его отличие только в отсутствии поддержки телефонных линей. Он работает только по цифровым линиям связи. В остальном он соответствует стандарту RS-232.
С полноценным RS-232 мы познакомимся позже.

6.4.2 SCI

Передача кадрами.
Формат кадра:

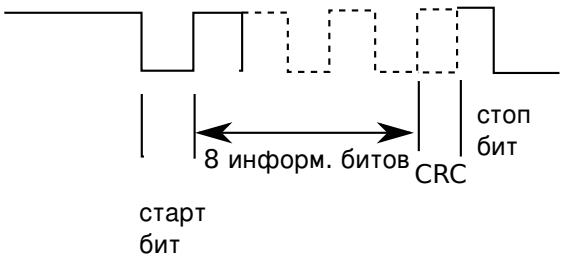


Рис. 6.5: Формат кадра

Передача идет побайтно. Первым идет нулевой. Потом идут информационные (высокий или низкий уровни), затем идет контрольный бит и стоп-бит.
Каждый кадр >= 0 бит или >=1 бит.
Данные кадра 10 или 11 бит.
Сужебные кадры: BREAK – все 0. IDLE – все 1

Структура SCI

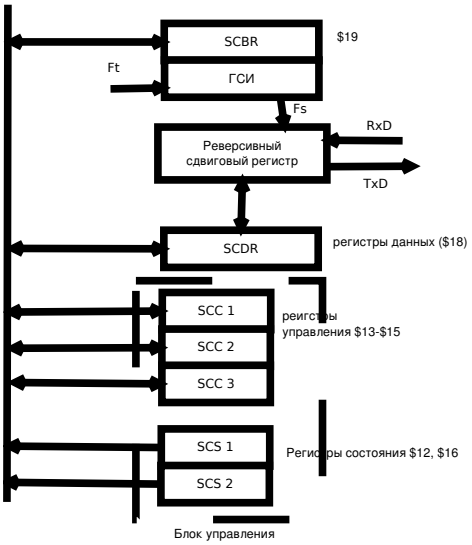


Рис. 6.6: Структура SCI

SCC 1,2,3 – регистры управление
SCS 1,2 – регистры состояния
ГСИ – генератор синхроимпульсов
Fs = Ft / Кд
Прием данных (чтение SCDR) – LDA $18
Режим приема – RxD. Режим передачи – TxD.
Передача данных (запись в SCDR) – STA $18
Передача обеспечивается блоком управления.

SCC1 Задает режим работы.

7	6	5	4	3	2	1	0

0 — PTY. Тип контроля – 0 четность, 1 – нечетность
1 — PEN – наличие контроля. 1 – есть. 0 – нет
4 — M – 0 - 10 бит, 1 – 11 бит
6 — ENSCI – Enable SCI. 1 – включить SCI.
7 — LOOPS. Режим контроля.

SCS1

Передача		Прием		Ошибки			
SCTE	TC	SCRF	IDLE	OR	NF	FE	PE

Передача После передачи в SCTE = 1; Данные из SCDR в сдвиговый регистр.
TC = 1 – конец передачи.

Прием данных SCRF = 1 – прием данных в SCDR.
IDLE = 1 – прием кадра IDLE. Нужно чтобы процессор мог подготовится к приему данных.

Ошибки приема OR = 1 — переполнение (приемник пропустил кадр и SCDR не считалось).
NF = 1 — шумы на линии.

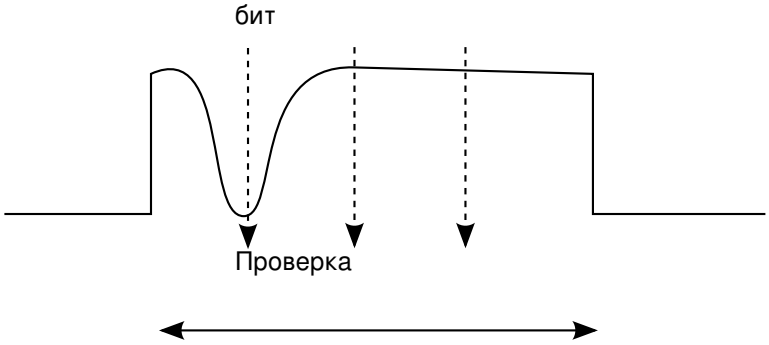


Рис. 6.7: Проверка NF

SCS2 RPF = 1 – начало приема
BKF = 1 – приход BREAK
Используется редко

SCC2, SCC3

SCC2

Разрешение прерываний				Режим работы			
SCTIE	TCIE	SCRIE	IDLIE	TE	RE	RWU	SBK

TE – режим приема (Transmit Enabled).
RE – пежим передачи (Receive Enabled).
RWU – Переход в режим ожидания
SBK – выдача BREAK
Первые 4 бита – разрешение прерывий на сответствующие биты в SCS1. Нужно, т.к. процессор может контролировать соотвтетсвующие события или читая биты или автоматически по прерыванию. IE – Interrupt Enabled.

Обслуживание SCI

1. По опросу:

   Чтение SCS1, SCS2 и так далее. В зависимости от состояния вызов соответтвующих управляющий функций.

2. По прерыванию:

   Если соответсвующий IE-бит = 1, процессор автоматически переходит к обработке прерываний.

SCC3

R8	T8	-	-	ORIE	NIE	FEIE	PEIE

R8 – контрольный бит приема. Приемный бит четности.
T8 – контрольный бит передачи. бит четности передачи.
Процессор должен записать бит четности в T8. Тогда он будет передан.
Последние 4 – прерывания.

Обслуживание прерываний

Вектора прерываний N_t – вектор обслуживания передачи.
N_r – вектор обслуживания приема.
N_l – вектор обслуживания ошибок.

Действия процессора

1. Сохранение основных регистров: A, X, CCR, PC.
2. Загрузка вектора прерываний в PC.

Обработчик прерываний

1. Определение прерываний.

   Чтение SCSI и анализ причины

2. выполнение обслуживания

   В зависимости от причины

Физические размеры Могут быть длинные линии.

SPI

SPI – Serial Perepherial Interface

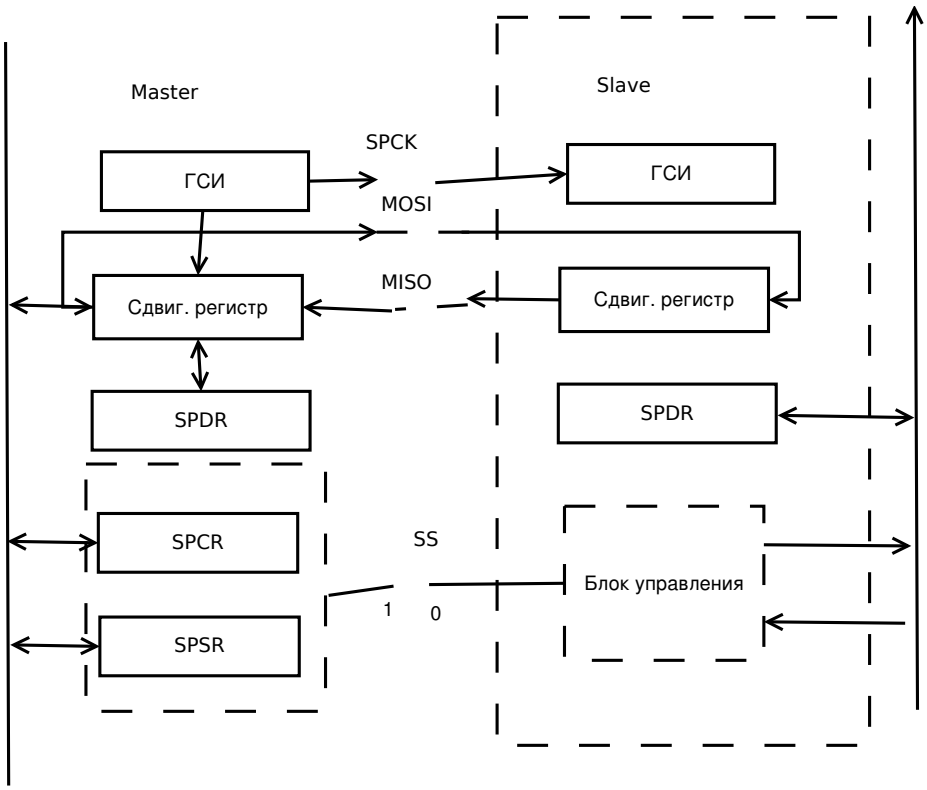


Рис. 6.8: Структура SPI

MOSI – Master Out, Slave IN
MISO – Master In, Slave Out.
SPCK – синхросигнал (Master)
SS – выбор устройства, master (1) или slave (0).
SPDR – регистр данных
SPCR – регистр управления ($10)
SPSCR – регистр состояния контроля($11)

SPSCR

SPCRF	ERRIE	OVRF	MODF	SPIE	MODFEN	SPR

SPCRF = 1 — после ввода данных в SPDR.
OVRF = 1 — переполнение (прием до чтения SPDR)
MODF = 1 — ошибка режима – SS=0 на Master.
SPTE = 1 — освобождение передатчика.
ERRIE =1 — перерывание при OVR или MODF.
SPR — Задают коэффициент деления (Кд) для ГСИ.

SPCR

SPIE	SPMST	CPOL	CPHA	CPWOM	SPE	SPTIE

SPIE = 1 – прерывание при SPCRF =1
SPTIE = 1 – прерывание при SPTE =1
SPMST = 1 – Mater
SPE =1 – включение SPI
CPOL – полярность.
SPHA – фронт синхроимпульса.

Обслуживание

1. По опросу
2. По прерыванию

Вектора прерываний V_t – ($FFE8) обслуживание передачи по SPTE = 1
V_r – ($FFEA) – прием или ошибка SPRF = 1 или ERRIE, MODF.

Физические размеры Близко расположенный объект. Обычно до 1.5м.

Использование

SCI и SPI используют выводы портов.
SCI: RxD -> PTE1.
TxD -> PTE0.

6.5 Таймерные модули

6.5.1 Общая структура

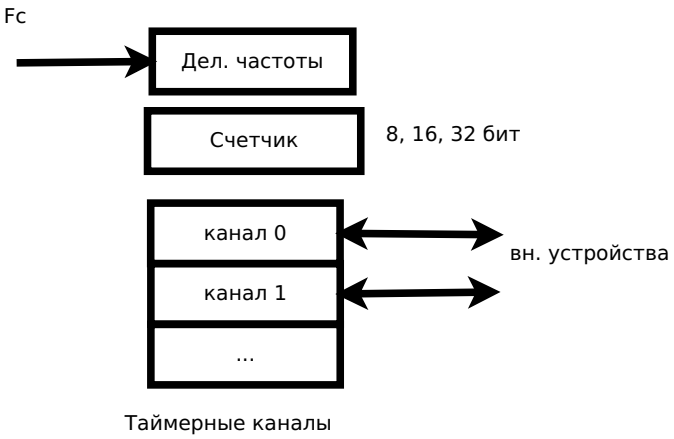


Рис. 6.9: Общуая структура

Функции: прием и выдача сигналов в заданные моменты времени. Работа в реальном времени.
$T_f = 1/F_t$.
Разрядности счетчиков: 8 – 256 T_c, 16 – 65536 T_c, 32 – 4294967296.
Число каналов: до 16. Число тайм-модулей.

6.5.2 Таймеры в микроконтроллере HC...

Основные сведения

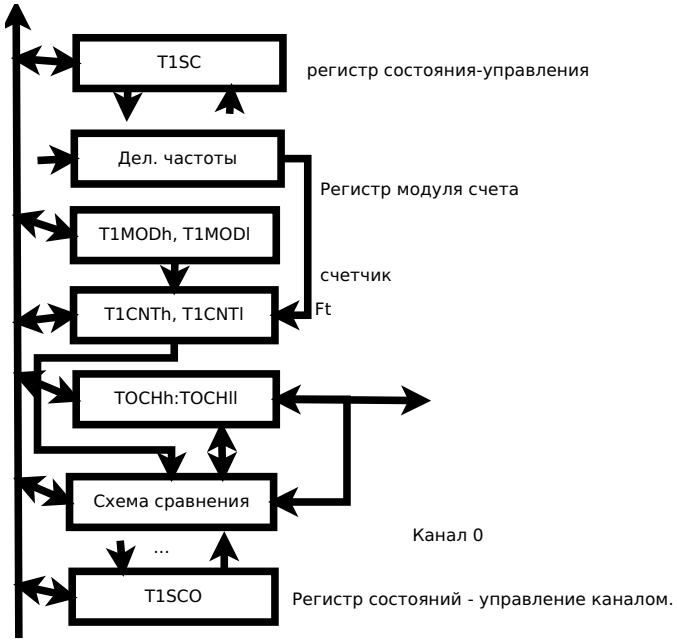


Рис. 6.10: Таймер

T1SC

TOF	TOIE	TSTOP	RST	-	PS2-0

TOF - переполнение. TOIE - прерывание. TSTOP - остановка. TRST - перезапуск PS2-0 - коэффициенты деления частоты.

Режим работы канала

1. Захват (Capture)

   По сигналу на входе T1CNT -> T1CHO.

   В этом режиме его можно использовать как частотомер: $T = \frac{T_2 - T_1}{T_t}$. Процессор считает T_1, T_2 и проводит вычитание и деление.

   Измерение частоты: считаем по фронту. Можно много полезного узнать о времени выполнения внешних событий.

2. Режим совпадение (Compore) Запись в T1CHO

3. Режим широтно-импульсной модуляции ШИМ. На выходе формуируем импульсы заданной частоты и длительности.

   T_n задается в T1CHO, T_t задается в T1MOD

T1SCO

CHF	CHIE	MS	ELS	TOV	CHMAX

CHF – флаг срабатывание канала (захват или совпадение)
CHIE – разрешение прерываний
MS: 00 - захват 01 - совпадение 1X совпадение с буферизацией.
ELS – сигнал: 01 – низкий-высокий. 10 - высокий-низкий, 11 - любой. 00 - вывод параллельного порта
TOV – разрешить широтноимпульсную модуляцию. Изменение сигнала T1CH при переполнении.
При частоте 1МГц, макс. время которое считает 65.5 мс. Если интересующий нас интервал времени больше, то нужно считать количество переполнение.

6.5.3 Сторожевой таймер (Watchdog)

Функция: предотвращение зависаний системы.
Причины: программные – бесконечный цикл. Аппаратные – отстутсвие отклика от внешних устройств.
Задает минимальное время за которое должен работать сторожевой таймер.

Сброс: программист должен предусмотреть специальную коммануд сброса. Если нет сброса выполняется reset. FFFF – адрес вектора reset во flash..

6.5.4 Аналого-цифровой преоразователь

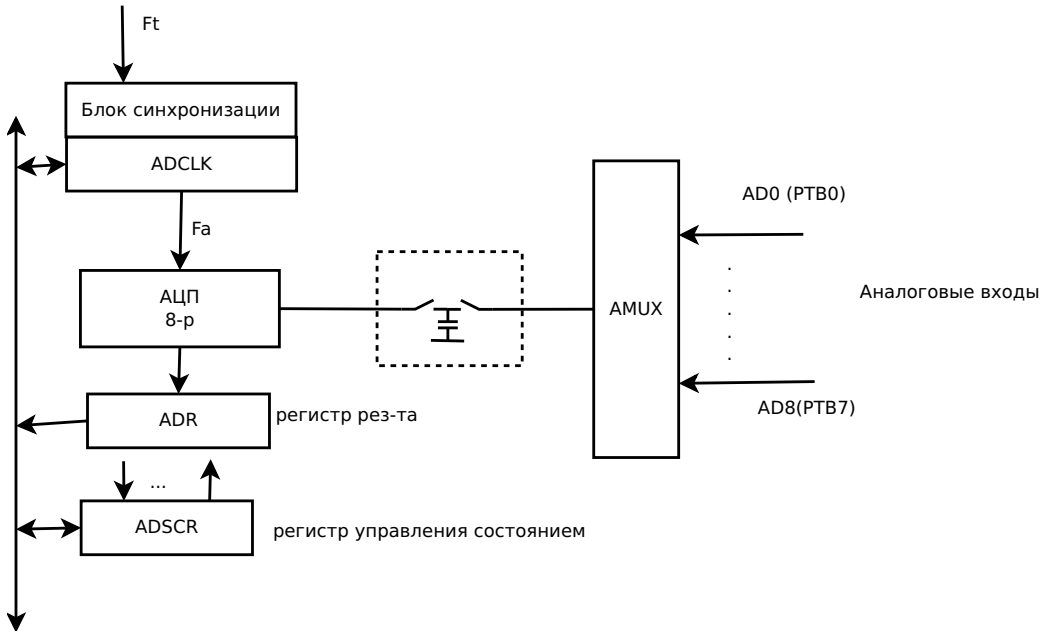


Рис. 6.11: АЦП

T1SCO

COCO	AIEN	ADCO	ADCH4-0

$F_a \sim 1MHz$

COCO — первый канал преобразования
AIEN — первый разряд преобразования
ADCO — режим. 0 – один повторяется один раз, 1 – непрерывный, меняется после каждого цикла преобразования
ADCH4-0 — выбор канала. 11111 – отключен. 00000-00111 – выбор номера канала. 11101 – V_r, 11110 – «земля». Обычно $V_r = E_{\text{п}}$. Диапазон измерения от 0 до $V_r = E_{\text{п}} = 5.0V$, точность $\frac{5V}{250} \simeq 20mV$

Цикл преобразования – 17 тактов. Время $\sim$ 17мкс. Отсюда видно, что частоты отслеживаеме порядка нескольких КГц.

Глава 7

Особенности архитектуры современных микропроцессоров

Цель:

- Повышение производительности
- расширение адресной памяти
- обеспечение надежности функционирования (защищенный режим)
- снижение энергопотербления (мобильность приложений

1 команда – 2-20 тактов.
Встает задача сделать, чтобы одна команда выполнялась за один такт. Возможно это? Да, возможно. Нужно сделать последовательный конвейер.

7.1 Конвейер команд

У современных процессоров от 3 до 30 ступений конвейера. Рассмотрим классический 6-и ступенчатый конвейер.
Введем следующие обозначения:
ВК – выборка команды
ДК – дешифрация команды
ФА – формирование адреса
ВО – выборка операнда
ВП – выполнение команды
ЗР – запись результата

T1SCO

К1	ВК	ДК	ФА	ВО	ВП	ЗР		
К2		ВК					ЗР	
К3			ВК	ДК	ФА	ВО	ВП	ЗР

ПР – простой. Команда не требует анной операции
ОЖ – ожидание. Требуется ждать предыдущего результата.

Пример: inc R2 mov (R2), R3 div R4,R3

T1SCO

К1	ВК	ДК	ПР	ПР	ВП	ПР				
К2		ВК	ДК	ОЖ	ОЖ	ФА	ВП	ПР		
К3			ВК	ДК	ПР	ПР	ОЖ	ОЖ	ВП	ПК

7.1.1 Эффективность работы конвейера

Можно использовать иксированный формат команд. Это характерно для RISC-процессора. Тип формата: 2 байта (МК) или 4 байта (МП).

7.2 Сокращение набора команд (RISC)

- исключение команд вызывающих простой процессора:

.

– деление

– вызов подпрограммы - адрес возврата сохраняется не в стеке, а в специальном регистре.
В таком случаи нет вложения подпрограмм.

- обработка ветвлений

  Ветвления тормозят конвейер. Поэтому реализуют предсказание ветвлений. Самый простой способ – считать, что программа пойдет по тому пути, по которому она уже шла раньше. Более сложный вариант - память предыдущих ветвлений по данному адресу. Если предсказание было неуспешным, приходится сбрасывать конвейер – убирать дешифрованные команды. Этот способ обычно довольно эффективен, но иногда можно сильно ошибится.

  Решается задача, где идут чередующиеся ветвления. В таком случаи, простейший случай будет всегда ошибаться.

  В таком случаи вводится память предыдущих ветвлений не только по адресу, но и по результату. Обычно около 4-х предыдущих результатов. На основе предыдущих результатов предсказывается последующий.

  За это отвечает специальный блок предсказания втвлений – Branch Prediction Unit.

  Практика показывает, что данные методы имеют эффективность порядка 90-95%.

- Статическое ветвление – специальная меткая команды ветвления, подсказывающая процессору более вероятный путь ветвления.

- Параллельное включение исполнительных устройств.

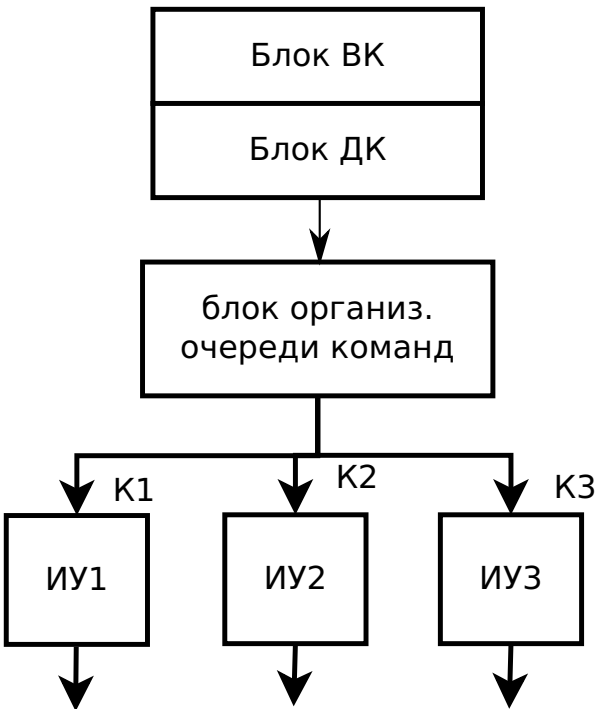


Рис. 7.1: ИУ1

Не все комманды можно выполнять одновременно, некоторые могут зависеть от результатов предыдущих, поэтому делают изменение порядка выполнение команд – внеочередное исполнение – Out-order Execution. Такая архитектура называется суперскалярной архитектурой

Возникает проблема – использование внутренних регистров. Делают специальные блоки, дублируют внутренние регистры. Все это приводит к усложнению схемы, повышению нагрева и энергопотребления.

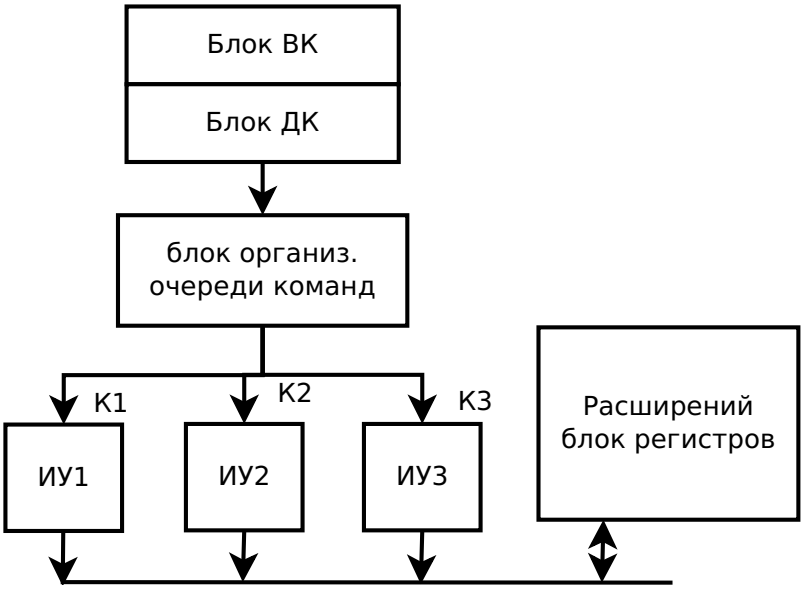


Рис. 7.2: ИУ1

- Ускорение обращения к памяти.
  Вводится промежуточная КЭШ-память.

7.3 Кэш-память

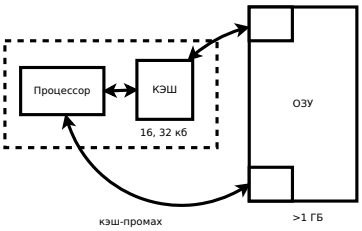


Рис. 7.3: КЭШ

кэш-попадание – необходимые данные есть в кэше.
кэш-промах – нужно обращаться к ОЗУ
Обычно используют множественно-ассоциативную организацию

Адрес:

31 12	11 5	4 0
Tag	Set	Byte

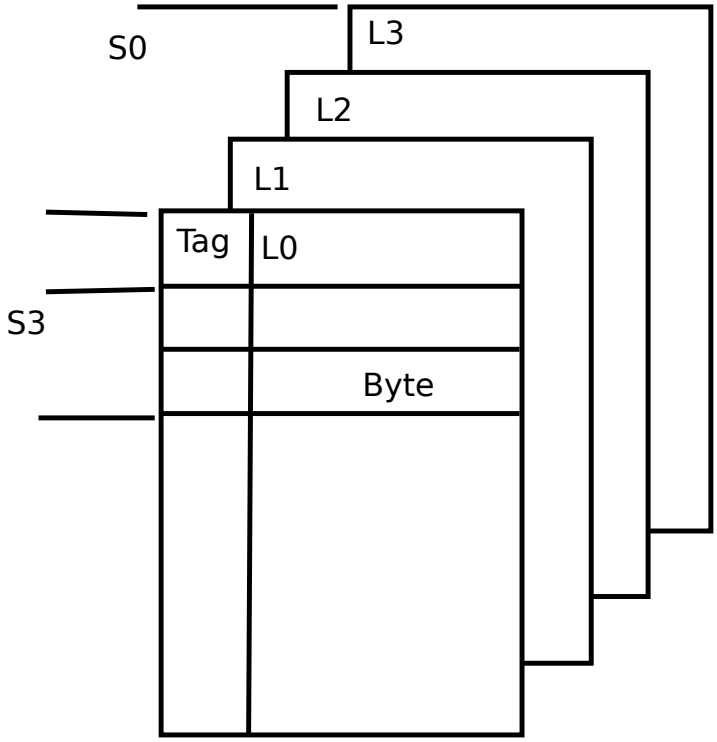


Рис. 7.4: КЭШ

Строка L – 16, 32, 64 байта.
128 наборов из 4 страниц. N блоков: 2, 4, 8. S – набор из строк. L0, L1, L2, L3.
Объём: 128x4x32 = 16кБайт.
Снаала выбирается номер набора [11:5] – Set
Потом сравнивается Tag[31:12] с Tag в наборе кэша
При совпадении – кэш-попадание.
Идут

Tag	Set
20 байт	32 байта данных

7.3.1 Заполнение

1. Запись тэга
2. Заполнение строки данных

   Часто используются пакетные циклы – заполнение порцией данных – пакетом. Например в pentium пакеты имеют размер 8 байт.

Нельзя гарантировать, что данные обязательно будут.

7.3.2 Кэш-промах при заполненном кэше

Одна из строк должна быть заменена на новую. Механизмы замены:

1. Случайный

   По случайному признаку. Самый простой. Обычно используют более интелектуальные признаки

2. Замена долго неиспользуемой строки.

   Считается, что строка, которая дольше всего не использовалась, больше не нужна.

   Механизм требует, чтобы мы хранили специальные биты - LRU – Least Recently Used, обычно 3 бита для 4-х строк: B0, B1, B2

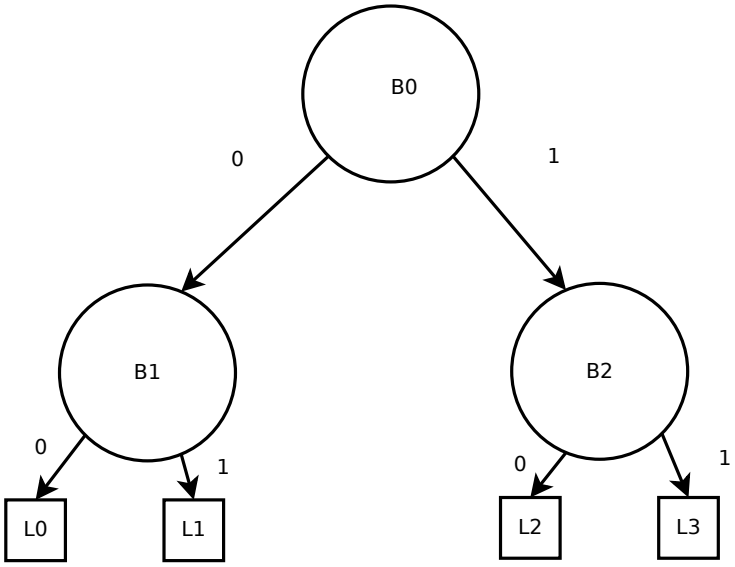


Рис. 7.5: Тэги

Анализ B0-B2 и выявление сроки для замены.

Служебная память для каждой строки:

V	M	B2-B0

V – бит присутствия (заполнение)
M – бит модификации (запись)
B2-B0 – биты LRU

7.3.3 Кэш-попадание при записи

Проблема: несоответствие ОЗУ и Кэша.
Решение проблемы:

1. Сквозная запись (Write-Through)

   Запись кэша в ОЗУ. Цена: высокая загрузка шины, снижение эффективности работы кэша.

2. Обратная запись (Write-Back)

   При кэш-попадании запись только в кэш. При замене модифицированной строки, чтобы не потерять данные, происходит автоматическая запись заменяемой строки в ОЗУ.

   Бывают системы, в которым допустимо обращение к ОЗУ нескольких устройств, например, многопроцессорные системы. Чтобы такая система работала, нужно использовать специальную процедуру – snooping. В этом случаи устройство может «подглядывать» в кэш-память и смотреть, изменились ли там строки и затем считать эту строку. В таком случаи имеется специальный протокол снупинга – довольно сложная вещь, который в рамках данного курса описываться не будет

Глава 8

Архитектура процессоров Intel

IA-32 – Intel Architecture, 32 bit (386, 486, Pentium, Core Duo)

8.1 Основные особенности

1. Поддержка сегментной адресации.
2. Реализация защиты памяти

Из-за пункта 2 следует два режима работы:

1. ральных адресов (Real) – при запуске. Очень ограничен.
2. защищенный (Protected)

8.2 Регистровая модель пользования.

Выделяется 2 модели работы:

- Пользователь (user) – прикладные задачи.
- Операционная система (Supervisor) – сложные задачи.

Рассмотрим первый режим.

8.2.1 Основные признаки

ZF – нуль
SF – знак
CF – перенос
OF – переполнение

AF – полуперенос
PF – четность
IF – маска прерываний
TF – трассировка – при установке в 1, процессор переходит в пошаговый режим. Режим отладки.
IOPL – защита устройств ввода-вывода
NT – вложение задач. Нужно для организации многозадачного режима

Сегментные регистры CS – сегменты команд
SS – сегмент стека
DS..GS – сгемнты данных

Сожержат 16 разрядный `селектор` сегмента. Они позволяют выбрать 64-х битный дескриптор с параметрами сегмента.

Дескриптор содержит:

- базовый адрес
- размер сегмента
- аттрибуты

Для каждого рабочего сгемента есть свой дескриптор. Таблица дескрипторов хранится в ОЗУ.

8.2.2 Обращение к сегменту

Тип обращения	сгемент регистр	Относительный адрес
Выборка команды	CS	EIP
Обращение к стеку	SS	ESP
Адрессация операнда	DS(CS, SS, ES, FS, GS) – обращение с указанием префикса. По-умолчанию к DS	EA

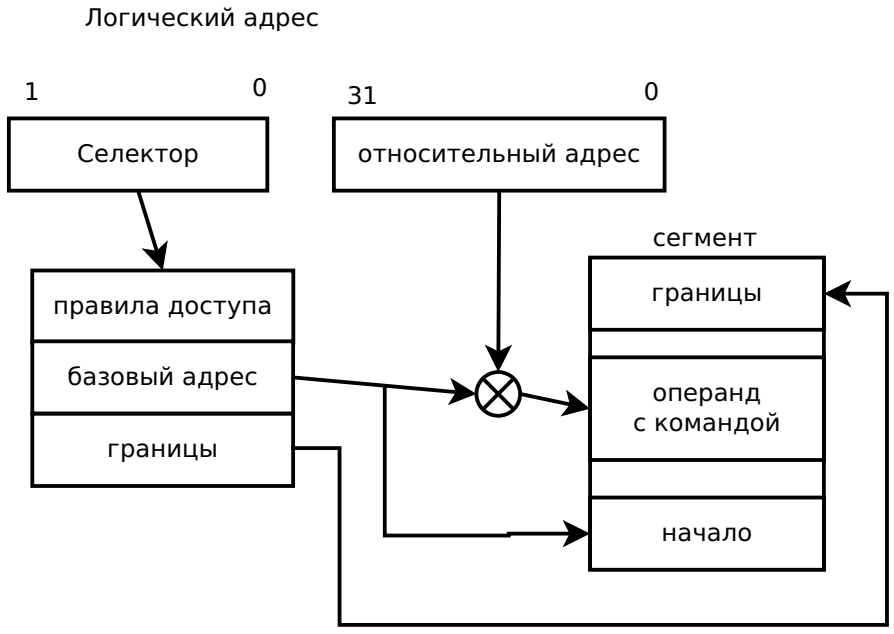


Рис. 8.1: Работа селектора

	OPC	MODR	SIB	DISP	IMM
префикс	байты адресации 1 или 2		смещение (1, 2 или 4 байта)	непосредственно операнд (1, 2 или 4 байта)	

от 1 до 12 байт

8.3 Способы адресации

Двухадресные команды:

1 операнд – регистр

2 операнд – регистр или ОЗУ

$ADDr, r/m$, где r - dst, r/m - операнд. $ADDmr$, где r - dst (назначение), r/m - операнд. $m + r \to m$

1. регистровая, EAX
2. косвенно-регистровая DS:[EBX] – сегмент выбоки DS.
   ES:[EBP] – префикс перед командой. Адрес = базовый адрес +
   Базовый адрс находится в дескрипторе, выбираемом селектором в ES.
3. косвенно-регистровая со смещением d.
   FS:[ECX+d], d – 8, 16, или 32 бита.
   Адрес = базовый адрес + (ECX) + d
4. Косвенно-еригстровая с индексированием.
   Индексный регистр – любой регистр общего назначения.
   $IR \to EAX, EBX$
   DS:[EBP + ECX], ECX – IR
   Адрес = базовй адрес (DS) + (EBP) + (ECX)
   Масштабирование IR – сдвиг влево на 1, 2 или 3 разряда. Если мы хотим сдвинуть, задается множитель квантования F:
   DS:[EBP + EDX*F], F = 1, 2, 4, 8 (нет сдвига, на 1 разряд, на два, на три)
5. прямая
   DS:[адрес]
6. непосредственная
   ADD EAX,100
   EAX + 100 → EAX
7. Относительная (в командах перехода) EIP + rel → EIP

8.3.1 Формат

Предикат, если не DS.
КоП – 1 байт.
Байты адресации:

1 байт, если не индексная.

2 байт, если индексная (2 регистра).

ES: [EAX]

ES: [EAX+EBP]

2-ой байт указывает IR или смещение.

Смещение - 1, 2, 4 байта

Непосредственный операнд: Im – 1,2, 4 байта.

8.4 Набор целочисленных команд

MOV	21, 22 2, m m, 2	(r2)-> (r1) (m) -> (r) r -> m	регистр->регистр память->регистр регистр->память
Обращение к стеку			
PUSH POP PUSHAD POPAD	r/m DS:[EAX+100] r/m	r/m -> стек стек -> r/m EAX, EMBX, ...EDI -> стек стек -> EDI, ..., EAX	

8.4.1 Команды ввода-вывода

Адрес ВУ -> непосредственный или регистр DX, 16 разр номер порта ВУ

IN	#	IN 100: port 100 -> EAX	
OUT	DX	EAX -> port(DX)	

8.4.2 Арифметиеские операции

SBC	r/m, r	r*r/m -> r/m	
	r/m, im	rm -> im -> r/m	
SBB	аналогично	r+r/m - CI -> r/m	
		Обращение к стеку	
CMP	r/m, r		r+m-r результат не сохраняется. Установка ZF, CF, DF, ...
	r/m, Im		
		Умножение	
MUL	EAX, r/m	(EAX)*(r/m) -> EDX:EAX - 2 байта	без знака
IMUL	EAX, r/m		
	r, r/m	(r)*(r/m) -> r	
	r, Im	(r)*(Im) -> r	Другие варианты не позволяют сохранить разряды, превышающие 32 бита
		Деление	
DIV	EAX, r/m	(EDX:EAX) / (r/m) -> EAX, остаток -> EDX	без знака. EDX:EAX - 64bit
IDIV	EAX, r/m	(EDX:EAX) / (r/m) -> EAX	со знаком
		Остальные	
INC	r/m	(r/m) + 1 -> r/m	
DEC	r/m	(r/m) - 1 -> r/m	
NEG	r/m	-(r/m) -> r/m	

8.4.3 Логические операции

AND	r, r/m		
	r/m, Im		
OR	r, r/m		
	r/m, Im		
XOR	r, r/m		
	r/m, Im		
NOR	r, r/m		
	r/m, Im		
		Логические сдвиги	
SHL	r/m, Im		сдвиг влево
	r/m, CL		
SHR	r/m, Im		сдвиг влево
	r/m, CL		
		Арифметические сдвиги	
SAR	r/m, Im		сдвиг влево
	r/m, CL		
SAL	r/m, Im		сдвиг влево
	r/m, CL		
		Циклические сдвиги	
ROL	r/m		сдвиг влево
ROR	r/m		сдвиг влево

8.4.4 Битовые операции

BT	r/m, r	bi -> CF	тестирование бита
	r/m, Im	bi -> CF	
BTS	r/m, r	bi -> CF, 1 -> bi	со сбросом
	r/m, Im	bi -> CF 1 -> bi	
BTR	r/m, r	bi -> CF, 0 -> bi	с инверсией
	r/m, Im	bi -> CF, 0 -> bi	

8.4.5 Команды обработки строк символов

Симол: b - байт, w - слово (2 байта), D - двойное слово (32 бита)

Адресация символов

Принята адресация: строка-источник

- Сгмент DS или другой (префикс)
- Индекс символа - ESI

строка-приемник:

- сегмент ES
- индекс EDI

Адрес символа (источник) = Баз адрес (DS или префикс) + (ESI)
(приемник) = Базовый адрес(ES) + (EDI)

Операции

		загрузка символов	
LODS(B,W,D)			B, W или D из строки-источника в AL, AX или EAX.
		загрузка символов	
STOS(B,W,D)			(AL, AX или EAX) -> строка -приемник, префикс повторения -> заполнение строки – испольвование REP STOS, ECX задает число повторяемых символов
MOVS(B,W,D)		(B,W, D) символ -> (B, W, D) приемник	можно с REP. переносит ECX символов
INS(B, W, D)		port DX -> строка-приемник	можно с REP, повторит ECX раз
OUTS(B, W, D)			вывод символов
		Сравнение строк	
CMPS(B,W,D)		BWDисточник -> BWDприемник	установка признаков в EFLAGS
REPE CMPS			Сравнение до равенства
REPNE CMPS			Находить разичия символов
SCAS(B,W,D)			сравнение строки-источника с EAX(AX, AL)
REPE SCAS			
REPNE SCAS			

8.4.6 Команды управления

Переходы: внутрисегментные (сегмент EIP) - NEAR, межсегмнатные (сегменты CS и EIP) - FAR
Адрес команды = базовый адрес(SS) + относительный адрес(EIP)

загрузка символов

JMP NEAR	rel	PC + rel -> PC	относительный
	r/m	r/m -> PC	указывается регистр или ячейка, откуда загружается
JMP FAR	prt1, ptr2 m DS:[EBP]	ptr1->CS, ptr2->EIP m_{16}->CS, m_{32} -> EIP	
CALL FAR	prt1, ptr2 m DS:[EBP]	CS, EIP -> стек, ptr1->CS, ptr2->EIP CS, EIP -> стек, m_{16}->CS, m_{32} -> EIP	
RET NEAR	Im	стек -> EIP, ESP + Im -> ESP	исключает из стека Im позиций
RET FAR	Im	стек -> CS, EIP, ESP + Im -> ESP	исключает из стека Im позиций

Условные переходы

Условие CC		CC..	сравнение без знака	CC	Сравнение со знаком
O NO	OF = 1 OF = 0	B	<	L	<
E NE	CF = 1 CF = 0	NB	≥	NL	≥
S NS	SF = 1 SF = 0	BE	≤	LE	≤
P NP	PF = 1 PF = 0	NBE	>	LNE	>

8.4.7 Организация циклов

ECX – счетчик циклов

LOOP	rel8	ECX - 1 -> ECX, если ECX ≠ 0, EIP+rel8 -> EIP	проверка в начале цикла
LOOPE	rel8	ECX - 1 -> ECX, если ECX ≠ 0 или ZF=1, EIP+rel8 -> EIP	
LOOPNE	rel8	ECX - 1 -> ECX, если ECX ≠ 0 или ZF=0, EIP+rel8 -> EIP	
JECX	rel8	ECX - 1 -> ECX, если ECX = 0 или ZF=1, EIP+rel8 -> EIP	проверка в конце цикла

8.4.8 Команды прерываний

INT	rel8	EFLAGS, CS, EIP -> стек; Vi -> CS, EIP	rel8 - номер вектора
INT3		EFLAGS, CS, EIP -> стек; V3 -> CS, EIP	вектор отладки
INT0		EFLAGS, CS, EIP -> стек; V4 -> CS, EIP, если OF=1	деление на ноль

8.4.9 Адресация стека

Адрес вершины = базовый адрес + ESP.

Выборка операнда

адрес = базовый адрес (DS) + (EAX) + 100
Затем ESP - 4 -> ESP

Запись в стек

адрес = базовый адрес (SS) + ESP
Пример:

- MOV EAX, EDX EDX -> EAX
- MOV DS:[EBP+10], EDX
  EDX->M EDX -> EAX. 1 байт. КОП + байт адресации + смещение -> 3 байта
- MOV ES:[EBP+EBX], 1000
  1000 – 2 байта. ES – префикс. EBX – IR
  префикс+КОП+2 байта адрес + 2 байта Im – 6 байт.

- POP r/m
  1. Выборка из стека
     адрес = базовый адрес(SS) + ESP
  2. Запись в регистр или ОЗУ
  3. ESP + 4 -> ESP
- BT EAX, 5
  b5 из EAX в CF

Новые примеры: Нужно сделать:

- 1. Формат команды, назначение байтов
     MOV EDX, DS:[ECX+(EBP*8)]
     1 байт – КОП. 2 байта адресация (индексная).
  2. Выборка команды:
     Адрес = азовый адрес + EIP.
     Базовый адрес хранится в дискрипторе, который выбирается селектором в CS.
  3. Адресация операндов.
     Адрес: базовый адрес + (ECX) + (EBP*8)
     Базовый адрес в дескрипторе, который выбирается в селекторе в DS.
  4. Команда заносит в EDX содержимое адресованной ячейки памяти.
- 1. Формат команды, назначение байтов
     DIV EAX, DS:[EBP+1000]
     1 байт – КОП. 3 байта адресация (байт адреса и два байта на смещение).
     Адрес операнда = Базовый адрес(DS) + EBP + 1000
  2. (EDX:EAX)/операнд -> EAX, остаток -> EDX
- 1. Формат команды, назначение байтов
     JMP FAR DS[100]
     межсегментный переход
     1 байт – КОП. 1 байт адреса.
     Адрес = Базовый адрес(DS) + 100
     Выбираем 2 байта -> CS
     (Адрес+2) -> Адрес -> EIP
- 1. Формат команды, назначение байтов
     CALL NEAR FS:[EBX+EBP]
     префикс + код операции + 2 байта адрес = 4 байта
     EIP -> стек.
     Адресация стека = Базовый адрес(SS) + ESP

8.5 Поддержка многозадачного режима

Выполнение нескольких задач в режиме разделения времени.

Каждая задача выполняется определенное время – тайм-слот. После этого ОС переключает процессор на другую задачу. ОС решает кому выделить больше времени, кому меньше.

Поскольку тайм-слот занимает какое-то время, то это квазиодновременное выполнение.

Сохранение контекста – информация о текущем состоянии задачи. Нужно сохранить содержимое основных регистров. Регистров много, поэтому в некоторых процессорах (Intel) это реализуется особенным способом.

8.5.1 Формирование сгемента состояния задачи (TTS – Task State Segment)

Процессоры Intel обеспечивают формирования сегмента состояния задачи – сегмента, хранящего контекст текущей задачи.

Сгемент содержит 104 байта – основная и дополнительная информация.

Сегмент довольно большой, побайтно его сохранять долго, поэтому обеспечивается автоматическое сохранение при переключении задач.

селектор возврата
SS0, ESP0
SS1, ESP1
SS2, ESP2
IIP, EFLAGS
EAX, ... EDI
CS, ..., GS
LDIR, CR3
служебная информация (БКВВ)

Все кроме служебной информации – 104 байта.

селектор возврата – селектор предыдущей задачи.

SS, ESP – состояние стеков уровней 0, 1, 2 (супервизор)

LDTR – селектор LDT, локальной таблицы которая используется в данной задаче.

CR3 – используется при табличной адресации.

БКВВ – битовая карта ввода вывода. Порт ввода-вывода n – бит n

8.5.2 Переключение задач

JMP FAR – выбор дескриптора TSS из GDI.

CALL FAR – то же самое.

Синхронизация дескриптора TSS.

Базовый адрес	граница TSS	TYPE	P	DPL	B

Базовый адрес – показывает где размещается TSS.

Граница – размер TSS

TYPE – показывает что это TSS

P – аттрибут присутствия

DPL – уровень защиты.

B – специфический атрибут. BUSY. Бит занятости. Занят – 1, 0 – свободен.

8.5.3 Правила обращения

Такие же как к сегменту данных. $DPL \geq (CPL; RPL)$

CALL – возможность обраения через шлюз к более защищенным TSS.

JMP FAR – без возврата к предыдущей (B=0)

CALL FAR – сохраняется селектор предыдущей задачи в TSS, B=1;

Таким образом, селектор сохраняется только по команде CALL. Из JMP возврата не предусмотрено, поэтому селектор не сохраняется.

8.5.4 Переключение

1. В TR записывается селектор из CS, вызывающей задачи (TSS)

   Если CALL, то предыдущий: TD в селектор возврата.

2. В регистре EFLAGS бит NT устанавливается в 1, если CALL (вложенные задачи)

   В дескрипторе старой задачи ставится B = 1.

3. Сохранение старого TSS.

   Переключение на новый TSS, загрузка регистров.

Завершение задачи – команда IRET. Если в EFLAGS NT=1 – возврат к предыдущей задаче (переключение на старый TSS используя селектор возврата).

Если вложения нет, востановливается CS, EIP, EFLAGS.

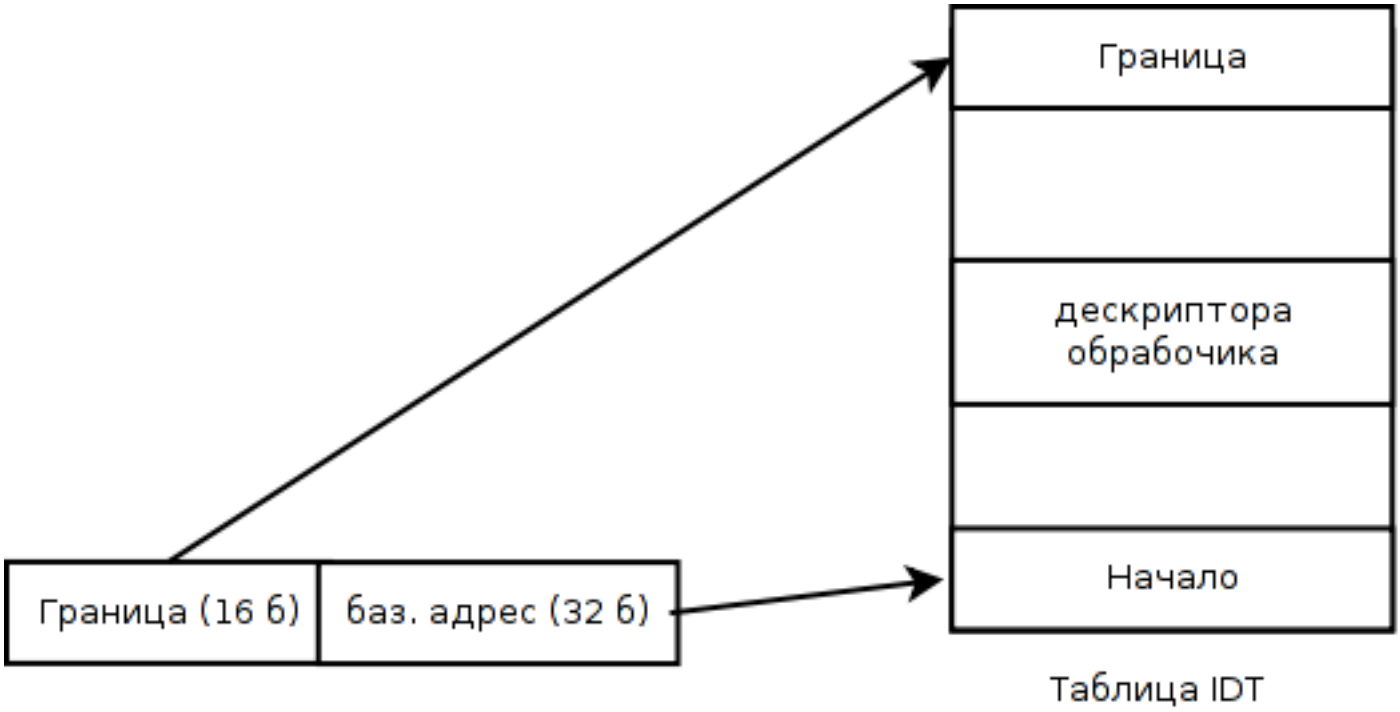


Рис. 8.2: Какая-то картинка

селектор	смещение	TYPE	P	GDPL

селектор – CS, смещение – EIP.

8.5.5 Обработка прерываний

Правила вызова: CPL $\leq$ GDPL
Желательно на уровне 0 (ядро ОС).

1. В стек уровня 0 загружаются EFLAGS, CS, ID
2. Устанавливаются IF = 1 (запрет прерываний) в EFLAGS
3. Выборка из IDT, загрузка CS, EIp
4. Выборка и выполнение первой команды обработки.

Обработчик прерываний должен заканчиваться коммандой IRET – возврат из обработки. IRET восстанавливает из стека EIP, CS, EFLAGS.

8.5.6 Типы прерываний

- 0 - 31 – служебные
- 32 – 255 – пользовательские.

8.5.7 Основные служебные прерывания

- 0 – деление на 0
- 1 – работа в пошаговом режиме (TF=1 в EFLAGS)
- 2 – сигнал NMI.
- 3 – INT3, программа отладки
- 4 – INT0, контроль переполнения
- 6 – неправильный код операции или неправильная адрессация
- 10 – недействительный TSS (P=0)
- 11 – отсутствие сегмента (P=0)

- 12 – ошибка обращения к стеку (параметры границы; P=0)
- 13 – нарушение защиты, нарушение границы стека (кроме SS)
  Нарушение границы GDT, LDT.
  Нарушение правил обращения (правил доступа)
  Превышение длины команды (5 айт)
- 14 – ошибка страничной адрессации.
- 16 – ошибка обработки чисел с плавающей точкой.