Выучить принципы Фон Неймана (глава 2 пособия В.Г.Баулы). Такое может быть в качестве вопроса

Далее привожу некоторые определения и вопросы-ответы, которые часто предлагаются на экзаменах (эталон ответа - в формулировке вашего лектора, какой он хочет видеть от вас ответ)

ПРЕРЫВАНИЯ. Система (аппарат) прерываний – возможность ЦП автоматически совершать переходы при возникновении определённых условий (*событий*). По *месту возникновения прерывания* делятся на *внутренние* (от ЦП и ОП) и *внешние* (от периферийных устройств). Прочитать главу 7 пособия В.Г.Баулы про прерывания.

Аппаратная реакция (аппаратные действия), выполняемые ЦП при поступлении непривилегированного (неприоритетного) сигнала прерывания. Найти в этой главе ответ на этот вопрос (4-5 предложений): Достаточно дать такой ответ:
1) Окончить выполнение текущей команды.
2) Если данное прерывание запрещено (замаскировано), то продолжить счёт текущей программы.
3) Сохранить минимальную информацию о выполняемой программе (флаги, CS, EIP записать в стек программы).
4) Установить запрет прерываний от внешних устройств.
5) Перейти на процедуру обработки данного прерывания.

Программная реакция на сигнал прерывания, выполняемая процедурой-обработчиком прерывания. Найти в этой главе ответ на этот вопрос (4-5 предложений).

МУЛЬТИПРОГРАММНЫЙ РЕЖИМ **Выучить !!!**
Мультипрограммный режим работы ЭВМ – режим работы ЭВМ, который позволяет находиться в оперативной памяти сразу нескольким (независимым друг от друга) программам, готовым к счету, переключение между выполнением которых происходит по определенному правилу.

Аппаратные средства, необходимые для поддержки работы в мультипрограммном режиме.
1) Система прерываний (для обеспечения автоматического переключения с одной программы на другую);
2) Механизм защиты памяти (см. ниже);
3) Аппарат привилегированных команд, т.е защищенный режим работы (см.ниже);
4) Таймер, т.е. встроенные в компьютер электронные часы (для переключения с одной программы на другую через определенные кванты времени).

2) Механизм защиты памяти – механизм, обеспечивающий безопасность одновременного нахождения в оперативной памяти нескольких независимых программ. Защита памяти гарантирует, что одна программа не сможет случайно или же предумышленно обратиться в память другой программы (по записи или чтению данных). Реализация. В ЦП имеются **2 внутренних регистра защиты** памяти, которые обозначаются условно как $A_{нач}$ и $A_{кон}$. Каждый из этих регистров может хранить любой физический адрес. Загрузчик, размещая программу в памяти, выделяет ей там непрерывный участок и заносит в регистры $A_{нач}$ и $A_{кон}$ начальные и конечные адреса этой области (команды загрузки значений на эти регистры должны быть привилегированными – иначе любая программа, перед обращением к памяти могла бы занести на эти регистры что угодно, например, адрес начала и конца оперативной памяти). А далее, перед каждым обращением этой программы к ОП по физическому адресу $A_{физ}$ автоматически (центральным процессором) проверяется *условие* $A_{нач} \le A_{физ} \le A_{кон}$. Если условие истинно, значит программа обращается в свою область памяти. Иначе – доступ в ОП не производится и возникает прерывание "нарушение защиты памяти". Назначение механизма защиты памяти: без него невозможна работа компьютера в *мультипрограммном режиме*.

3) Аппарат привилегированных команд. Компьютер имеет аппарат привилегированных команд, если все команды разбиты на 2 части: **привилегированные** команды (*запрещённые команды*) и

обычные команды (*команды пользователя*). ЦП такого компьютера может работать в двух режимах (привилегированном и пользовательском) и имеет **специальный регистр**, показывающий, в каком из этих двух режимов он сейчас работает. В привилегированном режиме ЦП может выполнять все команды; при попытке же выполнить привилегированную команду в пользовательском режиме, эта команда не выполняется и возникает прерывание. Назначение аппарата привилегированных команд: без этого аппарата невозможна работа компьютера в *мультипрограммном режиме*. Например, команды загрузки значений на **регистры защиты памяти** должны быть привилегированными – иначе любая программа, перед обращением к памяти могла бы занести на эти регистры что угодно, например, адрес начала и конца оперативной памяти. Привилегированными должны быть и команды обращения к разным внешним устройствам (дискам, принтеру…), чтобы не получилось, что сразу несколько программ одновременно работают с этим устройством. При необходимости обратиться к нужному внешнему устройству, программа обращается к служебным программам операционной системы, которые осуществят доступ к этому внешнему устройству, работая в привилегированном режиме. *Переключение* из обычного режима работы в привилегированный происходит автоматически (а не по какой-либо команде!!!) при обработке *процессором* сигнала прерывания, тем самым, *процедура-обработчик* прерывания начинает свою работу сразу в привилегированном режиме. Выход же из привилегированного режима в обычный осуществляется путем выполнения обычной (непривилегированной команде).

СПОСОБЫ ПОВЫШЕНИЯ ЭФФЕКТИВНОСТИ РАБОТЫ ПРОЦЕССОРА. Выучить !!!

1)Расслоение оперативной памяти (см. ниже).
2) Кэш-память (см. ниже).
3) Конвейер команд (см. ниже).

1) Расслоение оперативной памяти – это реализация оперативной памяти в виде нескольких параллельно работающих блоков – банков памяти**.** Обмен с банками можно вести параллельно, т.е. независимо друг от друга, и одновременно. Например, в одно и то же время можно считывать из 0-го и 2-го банка и записывать в 1-ый банк.
При этом обычно используется правило: ячейка с адресом *A* размещается в банке с номером *A mod m* (картинка). При этом байты памяти с *последовательными адресами* располагаются в *последовательных банках памяти* (так называемое “чередование” - interleaving). Расслоение необходимо, чтобы иметь возможность за одно обращение к памяти обмениваться не одним, а сразу несколькими байтами, расположенными в разных (как правило, последовательных) банках => за одно обращение к памяти можно читать сразу несколько команд или данных, расположенных в последовательных байтах памяти (а в реальных программах обычно и нужно вести такой обмен) => повышать скорость работы ЦП с памятью.

2) Кэш-память (cache – кошелёк, тайник, место для упрятывания вещей) – невидимая для программиста быстрая неадресуемая память относительно малого объема. Используется для *ускорения* работы ЦП с оперативной памятью
Идея. В каждый данный момент времени программа работает только с какой-то частью => с большой вероятностью она обратится к этим же ячейкам ОП (это так называемый ***принцип локальности***).
Суть принципа локальности (более подробно). Если было обращение к некоторой ячейке памяти, то, как правило, вскоре произойдёт новое обращение к той же ячейке (это т.н. *временная локальность*) или же вскоре произойдет обращение к близлежащим ячейкам (это т.н. *пространственная локальность*). Это экспериментально замеченный факт (теоретически ниоткуда не следует).
Пусть ЦП потребовалась ячейка с адресом А. Тогда содержимое этой ячейки *вместе с группой соседних ячеек* переписывается в КЭШ (меньше ОП примерно в 100-1000 раз, но быстрее её примерно в 10 раз), и только затем нужное слово считывается из КЭШа в ЦП. Если процессору снова потребуется какая-то ячейка, то он сначала проверит, нет ли ее в КЭШе (в силу принципа локальности эта ячейка скорее всего будет находиться в КЭШе). Если ячейка есть в КЭШе, то

процессор возьмет ее оттуда, если нет – прочитает ее из ОП в КЭШ *вместе с соседними ячейками*. Таким образом, процессор лишь изредка обращается за ячейками в ОП, в большинстве же случаев он берёт их из КЭШа. => Повышается скорость работы ЦП с ОП.
Однако имеет место некоторое замедление обмена с ОП, т.к. обращаемся к одной ячейке, а считывается целая группа ячеек. Но если при этом поддерживается расслоение памяти на m банков, то при параллельном считывании в КЭШ сразу m соседних ячеек *потерь времени почти не будет*.
Вопрос. Сформулируйте *принцип пространственной локальности*, на котором основано использование кэш-памяти (более развёрнуто, чем сказано выше). Обрабатываемые программой данные обычно располагаются в оперативной памяти компактными областями (массивы, локальные переменные процедуры, временные переменные при вычислении сложных выражений и т. д). При опережающем чтении последовательных ячеек памяти в кэш обеспечивается эффективная работа с такими данными.
Вопрос. Сформулируйте *принцип временной локальности*, на котором основано использование кэш-памяти (более развёрнуто, чем сказано выше). Доступ к некоторым обрабатываемым программой данным может производиться неоднократно в компактных частях программного кода (массивы, локальные переменные процедуры, временные переменные при вычислении сложных выражений и т. д). Даже если эти данные располагаются в удалённых друг от друга областях памяти, они накапливаются в кэш-памяти, что обеспечивает эффективную работу с такими данными.

Почему работа работа кэш памяти ЭВМ становится неэффективной при частых сигналах прерывания?
При поступлении сигнала прерывания процессор (чаще всего, если прерывание не замаскировано) переключается на выполнение другой программы (процедуры обработки прерывания), поэтому кэш память начинает заполняться командами и переменными другой программы, а команды и переменные старой программы вытесняются из кэш памяти. При продолжении счёта программы возникает промах кэш.

3) Конвейер команд – особенность архитектуры ЦП, когда он состоит из **нескольких блоков**, каждый из которых может выполнять один из шагов по выполнению команды. При использовании конвейера время выполнения одной команды не уменьшается. Выигрыш же от использования конвейера достигается за счет одновременного использования процессором нескольких команд.

Почему работа конвейерной ЭВМ замедляется при частых сигналах прерывания?
При поступлении сигнала прерывания процессор (чаще всего, если прерывание не замаскировано) переключается на выполнение другой программы (процедуры обработки прерывания), поэтому все не выполненные до конца команды удаляются с конвейера. При продолжении счёта программы они заново загружаются на конвейер.

Возможные случаи сбоя в работе *конвейера команд*:
а) *Сбой по данным*. Результат одной команды является операндом последующей команды. В этом случае 2 –ая команда ждёт, пока 1-ая не запишет свой результат (имеет место в приведённых задачах).
б) *Сбой по управлению*. При выполнении команды перехода. В этом случае конвейер сбрасывается (очищается) и на вход конвейера подаётся команда по адресу перехода.
в) *Сбой по аппаратуре*. Когда 2 команды одновременно пытаются получить доступ к устройству, которое в каждый момент может выполнять только 1 запрос. В этом случае доступ к устройству получает команда, попавшая в конвейер раньше, а 2-ая ждёт, пропуская свой этап выполнения (*например*, при **расслоении памяти** разрешён одновременный доступ к разным банкам памяти, но не одновременный доступ к одному и тому же банку).

Задача на конвейер команд. *Пусть в ЦП используется конвейерный способ выполнения команд, при котором параллельно выполняются следующие 6 этапов: 1) чтение команды, 2) дешифровка кода операции, 3) вычисление* $A_{исп}$,

*4) чтение операндов, 5) выполнение операции, 6) запись результата. Считая, что на каждый из этих этапов тратится 1 единица времени, и что вначале конвейер был свободен, определить, сколько времени будет затрачено на выполнение следующего фрагмента программы (*X - массив из двойных слов*):*

А)
```
sub EAX,3
add EBX,EAX
and X[EBX],EAX
```
Ответ: 13 ед. времени.

Р а с с у ж д е н и я:

Ед. вр.	*Чтение команды*	*Дешифровка КОП*	*Вычисление $A_{исп}$*	*Чтение операндов*	*Выполнение операции*	*Запись результата*
1	sub EAX,3					
2	add EBX,EAX	sub EAX,3				
3	and X[EBX],EAX	add EBX,EAX	sub EAX,3			
4		and X[EBX],EAX	add EBX,EAX	sub EAX,3		
5		and X[EBX],EAX	add EBX,EAX	Задержка по данным (EAX-?)	sub EAX,3	
6		and X[EBX],EAX	add EBX,EAX	Задержка по данным (EAX-?)		sub EAX,3
7		and X[EBX],EAX	Задержка по данным (EBX-?)	add EBX,EAX		
8		and X[EBX],EAX	Задержка по данным (EBX-?)		add EBX,EAX	
9		and X[EBX],EAX	Задержка по данным (EBX-?)			add EBX,EAX
10			and X[EBX],EAX			
11				and X[EBX],EAX		
12					and X[EBX],EAX	
13						and X[EBX],EAX

Б)
```
sub EAX,X[ESI]
add EBX,EAX
xor X[EDI],EBX
```
Ответ: 12 ед. времени.

Р а с с у ж д е н и я:

	Чтение команды	*Дешифровка КОП*	*Вычисление $A_{исп}$*	*Чтение операндов*	*Выполнение операции*	*Запись результата*
1	sub EAX,X[ESI]					
2	add EBX,EAX	sub EAX,X[ESI]				
3	xor X[EDI],EBX	add EBX,EAX	sub EAX,X[ESI]			
4		xor X[EDI],EBX	add EBX,EAX	sub EAX,X[ESI]		
5		xor X[EDI],EBX	add EBX,EAX	Задержка по данным (EAX-?)	sub EAX,X[ESI]	
6		xor X[EDI],EBX	add EBX,EAX	Задержка по данным (HAX-?)		sub EAX,X[ESI]
7			xor X[EDI],EBX	add EBX,EAX		
8			xor X[EDI],EBX	Задержка по данным (EBX-?)	add EBX,EAX	
9			xor X[EDI],EBX	Задержка по данным (EBX-?)		add EBX,EAX
10				xor X[EDI],EBX		
11					xor X[EDI],EBX	
12						xor X[EDI],EBX

ДОПОЛНИТЕЛЬНЫЕ ВОПРОСЫ ПО КОНВЕЙЕРУ КОМАНД

Что такое предсказание ветвления? - При выполнении на конвейере команды условного перехода, делается предположение о том, выполнит ли эта команда переход или продолжится последовательное выполнение программы. В соответствие с этим предположением на конвейер выбираются команды из наиболее вероятной (для данной команды условного перехода) ветви. Для чего оно используется? - Для сокращения простоев конвейера (т.е. для повышения эффективности работы конвейера).

Что такое спекулятивное выполнение? - При выполнении на конвейере команды условного перехода, на конвейер поступают (и выполняются) команды обеих ветвей (программы). Когда же становится известно, следует или нет выполнять переход, учитывается результат выполнения команд нужной ветви; результат выполнения команд второй ветви отбрасывается. Для чего оно используется? – Для сокращения простоев конвейера (для повышения эффективности работы конвейера).

Всякое разное.

Ответить на вопросы:

а) что может быть фактическим параметром макрокоманды;
б) что происходит с параметрами при макровызове.

а) любой текст (в том, числе пустой), сбалансированный по кавычкам и угловым скобкам, причём этот текст не должен содержать запятых и точек с запятой вне вышеуказанных кавычек и скобок.
б) при макровызове макропроцессор (=макрогенератор) находит макроопределение и подставляет в макрос фактические параметры вместо формальных, генерируя, таким образом, макрорасширение.
При ответе на этот пункт нужна ещё такая *добавка*: **при подстановке фактического параметра производится выполнение макрооператоров (&,<>,%,!)**

Объяснить, почему для **вывода** знаковых и беззнаковых чисел используются две разные макрокоманды **outint** и **outword**, в то время как для **ввода** целых чисел используется только одна макрокоманда **inint** ?

Во внутреннем машинном представлении знаковые и беззнаковые числа неразличимы, поэтому программисту при выводе необходимо указывать нужную макрокоманду (outint – интерпретирует содержимое ячейки как представление знакового числа в доп коде, outword – интерпретирует содержимое ячейки как представление беззнакового числа). В то же время во входном потоке (во внешнем текстовом представлении) знаковые и беззнаковые числа различаются. Причём при наличии знака минус число считается знаковым (и преобразуется макрокомандой inint в дополнительный код), в противном случае число считается беззнаковым (и записывается в память по правилам машинного представления чисел без знака)