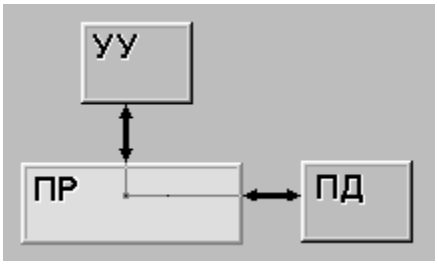
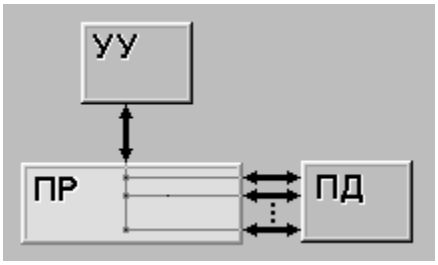
1. Классификации архитектур вычислительных систем. Подробнее о классификации Флинна.
    - https://www.parallel.ru/computers/taxonomy/
    - https://parallel.ru/computers/taxonomy/flynn.html
    - https://studref.com/313858/informatika/klassifikatsii_arhitektur_vychislitelnyh_sistem
    - http://scicenter.online/informatsionnyie-sistemy-scicenter/klassifikatsiya-arhitektur-vyichislitelnyih-64321.html

По-видимому, самой ранней и наиболее известной является классификация архитектур вычислительных систем, предложенная в 1966 году М.Флинном [1,2]. Классификация базируется на понятии *потока*, под которым понимается последовательность элементов, команд или данных, обрабатываемая процессором. На основе числа потоков команд и потоков данных Флинн выделяет четыре класса архитектур: SISD,MISD,SIMD,MIMD.

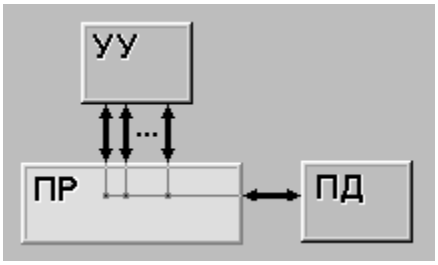


SISD (single instruction stream / single data stream) – одиночный поток команд и одиночный поток данных. К этому классу относятся, прежде всего, классические последовательные машины, или иначе, машины фон-неймановского типа, например, PDP-11 или VAX 11/780. В таких машинах есть только один поток команд, все команды обрабатываются последовательно друг за другом и каждая команда инициирует одну операцию с одним потоком данных. Не имеет значения тот факт, что для увеличения скорости обработки команд и скорости выполнения арифметических операций может применяться конвейерная обработка – как машина CDC 6600 со скалярными функциональными устройствами, так и CDC 7600 с конвейерными попадают в этот класс.

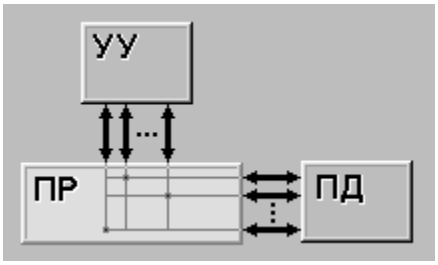


SIMD (single instruction stream / multiple data stream) – одиночный поток команд и множественный поток данных. В архитектурах подобного рода сохраняется один поток команд, включающий, в отличие от предыдущего класса, векторные команды. Это позволяет выполнять одну арифметическую операцию сразу над многими данными – элементами вектора. Способ выполнения векторных операций не оговаривается, поэтому обработка элементов вектора может производится либо процессорной матрицей, как в ILLIAC IV, либо с помощью конвейера, как, например,

в машине CRAY-1.



MISD (multiple instruction stream / single data stream) – множественный поток команд и одиночный поток данных. Определение подразумевает наличие в архитектуре многих процессоров, обрабатывающих один и тот же поток данных. Однако ни Флинн, ни другие специалисты в области архитектуры компьютеров до сих пор не смогли представить убедительный пример реально существующей вычислительной системы, построенной на данном принципе. Ряд исследователей [3,4,5] относят конвейерные машины к данному классу, однако это не нашло окончательного признания в научном сообществе. Будем считать, что пока данный класс пуст.



MIMD (multiple instruction stream / multiple data stream) – множественный поток команд и множественный поток данных. Этот класс предполагает, что в вычислительной системе есть несколько устройств обработки команд, объединенных в единый комплекс и работающих каждое со своим потоком команд и данных.

Итак, что же собой представляет каждый класс? В SISD, как уже говорилось, входят однопроцессорные последовательные компьютеры типа VAX 11/780. Однако, многими критиками подмечено, что в этот класс можно включить и векторно-конвейерные машины, если рассматривать вектор как одно неделимое данное для соответствующей команды. В таком случае в этот класс попадут и такие системы, как CRAY-1, CYBER 205, машины семейства FACOM VP и многие другие.

Бесспорными представителями класса SIMD считаются матрицы процессоров: ILLIAC IV, ICL DAP, Goodyear Aerospace MPP, Connection Machine 1 и т.п. В таких системах единое управляющее устройство контролирует множество процессорных элементов. Каждый процессорный элемент получает от устройства управления в каждый фиксированный момент времени одинаковую команду и выполняет ее над своими локальными данными. Для классических процессорных матриц никаких вопросов не возникает, однако в этот же класс можно включить и векторно-конвейерные машины, например, CRAY-1. В этом случае каждый элемент вектора надо рассматривать как отдельный элемент потока данных.

Класс MIMD чрезвычайно широк, поскольку включает в себя всевозможные мультипроцессорные системы: Cm*, C.mmp, CRAY Y-MP, Denelcor HEP,BBN Butterfly, Intel Paragon, CRAY T3D и многие другие. Интересно то, что если конвейерную обработку рассматривать как выполнение множества команд (операций ступеней конвейера) не над одиночным векторным потоком данных, а над множественным скалярным потоком, то все рассмотренные выше векторно-конвейерные компьютеры можно расположить и в данном классе.

Достоинства и недостатки. Предложенная схема классификации вплоть до настоящего времени является самой применяемой при начальной характеристике того или иного компьютера. Если говорится, что компьютер принадлежит классу SIMD или MIMD, то сразу становится понятным базовый принцип его работы, и в некоторых случаях этого бывает достаточно. Однако видны и явные недостатки. В частности, некоторые заслуживающие внимания архитектуры, например dataflow и векторно--конвейерные машины, четко не вписываются в данную классификацию. Другой недостаток - это чрезмерная заполненность класса MIMD. Необходимо средство, более избирательно систематизирующее архитектуры, которые по Флинну попадают в один класс, но совершенно различны по числу процессоров, природе и топологии связи между ними, по способу организации памяти и, конечно же, по технологии программирования.

Наличие пустого класса (MISD) не стоит считать недостатком схемы. Такие классы, по мнению некоторых исследователей в области классификации архитектур [6,7], могут стать чрезвычайно полезными для разработки принципиально новых концепций в теории и практике построения вычислительных систем.

2. Параллелизм на уровне битов, инструкций, данных, потоков.
    - https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B
    - https://helpiks.org/6-83955.html

Параллельные вычислительные системы — это физические компьютерные, а также программные системы, реализующие тем или иным способом параллельную обработку данных на многих вычислительных узлах.

Параллелизм на уровне битов

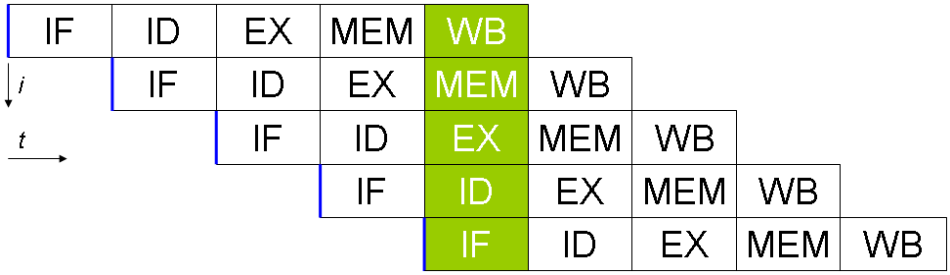
Эта форма параллелизма основана на увеличении размера машинного слова. Увеличение размера машинного слова уменьшает количество операций, необходимых процессору для выполнения действий над переменными, чей размер превышает размер машинного слова. К примеру: на 8-битном процессоре нужно сложить два 16-битных целых числа. Для

этого вначале нужно сложить нижние 8 бит чисел, затем сложить верхние 8 бит и к результату их сложения прибавить значение флага переноса. Итого 3 инструкции. С 16-битным процессором можно выполнить эту операцию одной инструкцией.

Исторически 4-битные микропроцессоры были заменены 8-битными, затем появились 16-битные и 32-битные. 32-битные процессоры долгое время были стандартом в повседневных вычислениях. С появлением технологии x86-64 для этих целей стали использовать 64-битные процессоры.

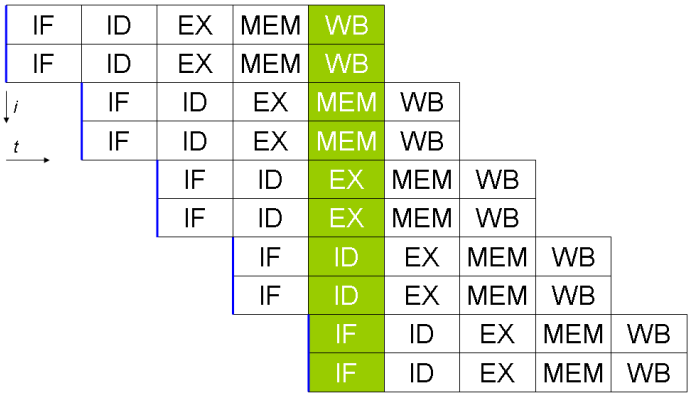
Параллелизм на уровне инструкций

Компьютерная программа — это, по существу, поток инструкций, выполняемых процессором. Но можно изменить порядок этих инструкций, распределить их по группам, которые будут выполняться параллельно, без изменения результата работы всей программы. Данный приём известен как параллелизм на уровне инструкций. Продвижения в развитии параллелизма на уровне инструкций в архитектуре компьютеров происходили с середины 1980-х до середины 1990-х.



Классический пример пятиступенчатого конвейера на RISC-машине (IF = выборка инструкции, ID = декодирование инструкции, EX = выполнение инструкции, MEM = доступ к памяти, WB = запись результата в регистры).

Современные процессоры имеют многоступенчатый конвейер команд. Каждой ступени конвейера соответствует определённое действие, выполняемое процессором в этой инструкции на этом этапе. Процессор с N ступенями конвейера может иметь одновременно до N различных инструкций на разном уровне законченности. Классический пример процессора с конвейером — это RISC-процессор с 5-ю ступенями: выборка инструкции из памяти (IF), декодирование инструкции (ID), выполнение инструкции (EX), доступ к памяти (MEM), запись результата в регистры (WB). Процессор Pentium 4 имеет конвейер в 31 ступень[5].



Пятиступенчатый конвейер суперскалярного процессора, способный выполнять две инструкции за цикл. Может иметь по две инструкции на каждой ступени конвейера, максимум 10 инструкций могут выполняться одновременно.

Некоторые процессоры, дополнительно к использованию конвейеров, обладают возможностью выполнять несколько инструкций одновременно, что даёт дополнительный параллелизм на уровне инструкций. Возможна реализация данного метода при помощи суперскалярности, когда инструкции могут быть сгруппированы вместе для параллельного выполнения (если в них нет зависимости между данными (завимости по данным)). Также возможны реализации с использованием явного параллелизма на уровне инструкций: VLIW и EPIC.

Параллелизм данных

Основная идея подхода, основанного на параллелизме данных, заключается в том, что одна операция выполняется сразу над всеми элементами массива данных. Различные фрагменты такого массива обрабатываются на векторном процессоре или на разных процессорах параллельной машины. Распределением данных между процессорами занимается программа. Векторизация или распараллеливание в этом случае чаще всего выполняется уже на этапе компиляции — перевода исходного текста программы в машинные команды. Роль программиста в этом случае обычно сводится к заданию настроек векторной или параллельной оптимизации компилятору, директив параллельной компиляции, использованию специализированных языков для параллельных вычислений.

Параллелизм задач

Стиль программирования, основанный на параллелизме задач, подразумевает, что вычислительная задача разбивается на несколько относительно самостоятельных подзадач и каждый процессор загружается своей собственной подзадачей.

3. Проблемы параллельных вычислений, закон Амдала и закон Густафсона-Барсиса.
   - http://www.hpcc.unn.ru/files/HTML_Version/introduction.html
   - https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BA%D0%BE%D0%BD_%D0%93%D1%83%D1%81%D1%82%D0%B0%D0%B2%D1%81%D0%BE%D0%BD%D0%B0_%E2%80%94_%D0%91%D0%B0%D1%80%D1%81%D0%B8%D1%81%D0%B0
   - http://ssd.sscc.ru/ru/content/%D0%B7%D0%B0%D0%BA%D0%BE%D0%BD-%D0%B0%D0%BC%D0%B4%D0%B0%D0%BB%D0%B0

Можно отметить также, что ряд исследователей высказывают свои сомнения о возможности широкого практического применения параллельных вычислений в силу следующего набора причин:

- **высокая стоимость параллельных систем** - в соответствии с подтверждаемым на практике законом Гроша (Grosch), производительность компьютера возрастает пропорционально квадрату его стоимости и, как результат, гораздо выгоднее получить требуемую вычислительную мощность приобретением одного производительного процессора, чем использование нескольких менее быстродействующих процессоров.
  Для ответа на данное замечание следует учесть, что рост быстродействия последовательных ЭВМ не может продолжаться бесконечно; кроме того, компьютеры подвержены быстрому моральному старению и результативность наиболее мощного процессора на данный момент времени быстро перекрывается новыми моделями компьютеров (а обновление морально устаревшего компьютера требует, как правило, больших финансовых трат);
- **потери производительности для организации параллелизма** - согласно гипотезе Минского (Minsky), ускорение, достигаемое при использовании параллельной системы, пропорционально двоичному логарифму от числа процессоров (т.е. при 1000 процессорах возможное ускорение оказывается равным 10).
  Безусловно, подобная оценка ускорения может иметь место при распараллеливании определенных алгоритмов. Вместе с тем, существует большое количество задач, при параллельном решении которых достигается 100% использование всех имеющихся процессоров параллельной вычислительной системы;
- **постоянное совершенствование последовательных компьютеров**
  - в соответствии с широко известным законом Мура (Moore) мощность последовательных процессоров возрастает практически в 2 раза каждые 18-24 месяцев (исторический экскурс показывает, что быстродействие ЭВМ увеличивалось на порядок каждые 5 лет) и, как результат, необходимая производительность может быть достигнута и на "обычных" последовательных компьютерах.
  Данное замечание близко по сути к первому из приведенных здесь возражений против параллелизма. В дополнение к имеющемуся ответу на это замечание можно отметить, что аналогичное развитие свойственно и параллельным системам. Кроме того, применение параллелизма позволяет получать желаемое ускорение вычислений без какого-либо ожидания новых более быстродействующих процессоров;
- **существование последовательных вычислений** - в соответствии с законом Амдала, ускорение процесса вычислений при использовании p процессоров ограничивается величиной $S_p \leq \frac{1}{f+\frac{1-f}{p}}$, где f есть доля последовательных вычислений в применяемом алгоритме обработки данных (т.е., например, при наличии всего 10% последовательных команд в выполняемых вычислениях, эффект использования параллелизма не может превышать 10-кратного ускорения обработки данных).
  Данное замечание характеризует одну из самых серьезных проблем в области параллельного программирования (алгоритмов без определенной доли последовательных команд практически не существует). Однако часто доля последовательных действий характеризует не возможность параллельного решения задач, а последовательные свойства применяемых алгоритмов. Как результат, доля

последовательных вычислений может быть существенно снижена при выборе более подходящих для распараллеливания алгоритмов;

- **зависимость эффективности параллелизма от учета характерных свойств параллельных систем** - в отличие от единственности классической схемы фон Неймана последовательных ЭВМ параллельные системы отличаются существенным разнообразием архитектурных принципов построения, и максимальный эффект от использования параллелизма может быть получен только при полном использовании всех особенностей аппаратуры - как результат, перенос параллельных алгоритмов и программ между разными типами систем становится затруднительным (если вообще возможен).

  Для ответа на данное замечание следует отметить, что "однородность" последовательных ЭВМ также является кажущейся и их эффективное использование тоже требует учета свойств аппаратуры. С другой стороны, при всем разнообразии архитектур параллельных систем, тем не менее, существуют и определенные "устоявшиеся" способы обеспечения параллелизма (конвейерные вычисления, многопроцессорные системы и т.п.). Кроме того, инвариантность создаваемого программного обеспечения может быть обеспечена и при помощи использования типовых программных средств поддержки параллельных вычислений (типа программных библиотек MPI, PVM и др.);
- **существующее программное обеспечение ориентировано в основном на последовательные ЭВМ** - данное возражение состоит в том, что для большого количества задач уже имеется подготовленное программное обеспечение и все эти программы ориентированы главным образом на последовательные ЭВМ - как результат, переработка такого количества программ для параллельных систем вряд ли окажется возможным.

Проблематика параллельных вычислений является чрезвычайно широкой областью теоретических исследований и практически выполняемых работ и обычно подразделяется на следующие направления деятельности:

- разработка параллельных вычислительных систем
- анализ эффективности параллельных вычислений
- создание и развитие параллельных алгоритмов для решения прикладных задач в разных областях практических
- разработка параллельных программных систем
- создание и развитие системного программного обеспечения для параллельных вычислительных систем

Закон Амдала

Одной из главных характеристик параллельных систем является ускорение S параллельной системы, которое определяется выражением: $S_p = \frac{T_1}{T_p}$, где T_1 − время решения задачи на однопроцессорной системе, а T_p − время решения той же задачи на p − процессорной системе.

Пусть $F = F_{\text{ск}} + F_{\text{пр}}$, где F− общее число операций в задаче, $F_{\text{пр}}$ − число операций, которые можно выполнять параллельно, а $F_{\text{ск}}$ − число скалярных (не распараллеливаемых) операций. Если

доля f от общего объёма вычислений может быть получена только последовательными расчётами, тогда получаем известный закон Амдала $S_p \leq \frac{1}{f+\frac{1-f}{p}} \leq S^* = \frac{1}{f}$

Закон Амдала характеризует одну из самых серьезных проблем параллельного программирования (алгоритмов без определенной доли последовательных команд практически не существует).

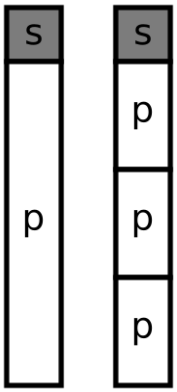
Следствия из закона Амдала:

- Ускорение зависит от потенциального параллелизма задачи (величина f) и параметров

аппаратуры (числа процессоров p).

- Предельное ускорение определяется свойствами задачи. Например, при наличии всего 10% последовательных команд в выполняемых вычислениях, эффект использования параллелизма не может превышать 10-кратного ускорения обработки данных.

Закон Густафсона (иногда Густавсона) — Барсиса (англ. Gustafson – Barsis's law) — оценка максимально достижимого ускорения выполнения параллельной программы, в зависимости от количества одновременно выполняемых потоков вычислений («процессоров») и доли последовательных расчётов. Аналог закона Амдала: Джон Густафсон (англ. John L. Gustafson) и Эдвин Барсис (Edwin H. Barsis) представили статью «Переоценка закона Амдала» в 1988 году.



Закон Амдала предполагает, что объем задачи остается неизменным. Задача слева обрабатывается одним потоком, задача справа — тремя. Сравнивается время выполнения.

Закон Густафсона — Барсиса выражается формулой:

$S_n = s + (1 - s)n = n + (1 - n)s$, где

s — доля последовательных расчётов в программе,
n — количество процессоров.

Данную оценку ускорения называют ускорением масштабирования (англ. scaled speedup), так как данная характеристика показывает, насколько эффективно могут быть организованы параллельные вычисления при увеличении сложности решаемых задач.

Отличие от закона Амдала

При оценке ускорения параллельного выполнения закон Амдала предполагает, что объем задачи остается постоянным. Величина ускорения по закону Амдала показывает, во сколько раз меньше времени потребуется параллельной программе для выполнения. Однако величину ускорения можно рассматривать и как увеличение объема выполненной задачи за постоянный промежуток времени. Закон Густафсона появился именно из этого предположения.

Вывод формулы

Ускорение масштабирования определяется как отношение объема вычислений, выполненных с использованием многопоточности, к объему вычислений, выполненных последовательно за один и тот же промежуток времени.

$$S_n = \frac{p \cdot n + s}{p + s} = \frac{n(p+s)}{p+s} + \frac{s - s \cdot n}{p+s} = n + (1-n)\frac{s}{p+s} = n + (1-n)s,$$ где

s — доля последовательных расчётов в программе,

p — доля параллельных расчётов в программе,

n — количество процессоров.

4. Конвейерная обработка инструкций. Скалярная и суперскалярная организация вычислений.
   - https://sites.google.com/site/arhitekturaevm1/razdel-2-arhitektura-i-struktura-vycislitelnyh-masin-i-sistem/4-konvejernaa-obrabotka-komand-superskalarizacia
   - https://studopedia.ru/10_201696_RISC-arhitekturi-konveyernaya-obrabotka-instruktsiy.html
   - https://studfiles.net/preview/5172071/
   - https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%BF%D0%B5%D1%80%D1%81%D0%BA%D0%B0%D0%BB%D1%8F%D1%80%D0%BD%D0%BE%D1%81%D1%82%D1%8C
   - https://studopedia.ru/12_219982_printsipi-skalyarnoy-i-superskalyarnoy-obrabotki-dannih.html

Pipelining

Конвейерная обработка - способ выполнения команд процессором, при котором выполнение следующей команды начинается до полного окончания выполнения предыдущей команды (в предположении отсутствия ветвления).

Возможность конвейерной обработки связана с разделением процесса выполнения команд на *последовательные этапы*: выборки команды, дешифровки, выборки операндов, выполнения команды, запись результата в память.

Особенности архитектуры суперскалярных процессоров

Термин «суперскалярная архитектура» обозначает процессорную архитектуру, которая содержит более одного вычислительного блока. Эти вычислительные блоки, или конвейеры, являются узлами, где происходят все основные процессы обработки данных и команд.

Суперскалярная архитектура является развитием скалярной архитектуры SISD (Single Instruction Single Date) - один поток команд управляет одним потоком данных.

Как известно, производительность любого процессора при выполнении заданной программы зависит от трех параметров: такта (или частоты) синхронизации, среднего количества команд, выполняемых за один такт, и общего количества выполняемых в программе команд. Невозможно изменить ни один из указанных параметров независимо от других, поскольку соответствующие базовые технологии взаимосвязаны: частота синхронизации определяется достигнутым уровнем технологии интегральных схем и функциональной организацией процессора, среднее количество тактов на команду зависит от функциональной организации и архитектуры системы команд, а количество выполняемых в программе команд определяется архитектурой системы команд и технологией компиляторов.

Конвейерная обработка команд. Риски.

Выполнение каждой команды складывается из ряда последовательных этапов (шагов, стадий), суть которых не меняется от команды к команде. С целью увеличения быстродействия процессора и максимального использования всех его возможностей в современных микропроцессорах используется конвейерный принцип обработки информации. Этот принцип подразумевает, что в каждый момент времени процессор работает над различными стадиями выполнения нескольких команд, причем на выполнение каждой стадии выделяются отдельные аппаратные ресурсы. По очередному тактовому импульсу каждая команда в конвейере продвигается на следующую стадию обработки, выполненная команда покидает конвейер, а новая поступает в него.

В различных процессорах количество и суть этапов различаются. Рассмотрим принципы конвейерной обработки информации на примере пятиступенчатого конвейера, в котором выполнение команды складывается из следующих этапов:

1.IF (Instruction Fetch) - считывание команды в процессор;

2.ID (Instruction Decoding) - декодирование команды;

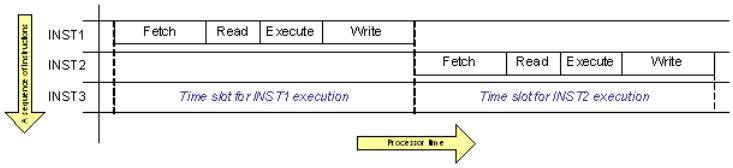
3.OR (Operand Reading) - считывание операндов;

4.EX (Executing) - выполнение команды;

5.WB (Write Back) - запись результата.

Пример:

Диаграмма исполнения инструкций для архитектуры Фон Неймана:



Pipelining: можно выделить 4 основных этапа исполнения инструкций:

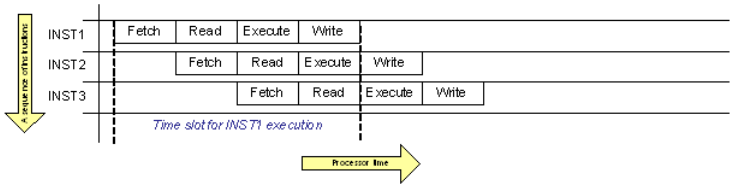
1) Fetch stage – извлечение и декодирование.

2) Read stage – чтение операндов.

3) Execute stage – исполнение.

4) Write stage – запись результата.

Если все стадии равновелики по времени исполнения, то возможна конвейерная организация :



Так как в каждом такте могут выполняться различные стадии обработки команд, то длительность такта выбирается исходя из максимального времени выполнения всех стадий. Кроме того, следует учитывать, что для передачи команды с одной стадии на другую требуется определенное дополнительное время (t), связанное с записью промежуточных результатов обработки в буферные регистры.

Значительное преимущество конвейерной обработки перед последовательной имеет место в идеальном конвейере, в котором отсутствуют конфликты и все команды выполняются друг за другом без перезагрузки конвейера. Наличие конфликтов снижает реальную производительность конвейера по сравнению с идеальным случаем.

Конфликты - это такие ситуации в конвейерной обработке, которые препятствуют выполнению очередной команды в предназначенном для нее такте.

Конфликты делятся на три группы:

·структурные,

·по управлению,

·по данным.

Структурные конфликты возникают в том случае, когда аппаратные средства процессора не могут поддерживать все возможные комбинации команд в режиме одновременного выполнения с совмещением.

Конфликты по управлению возникают при конвейеризации команд переходов и других команд, изменяющих значение счетчика команд.

Конфликты по данным возникают в случаях, когда выполнение одной команды зависит от результата выполнения предыдущей команды.