Конвейер команд

Конвейеризация вычислений

Наиболее важный архитектурный прием повышения производительности – конвейеризация вычислений.
Конвейерная обработка в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями (стадиями) конвейера и выделении для каждой из них отдельного блока аппаратуры.
Конвейеризацию используют для повышения производительности таких устройств как операционные устройства, оперативная память.
Наибольший эффект достигается при конвейеризации машинного цикла – этапов выполнения команд. Ступени разделяются с помощью буферных регистров или буферной памяти.

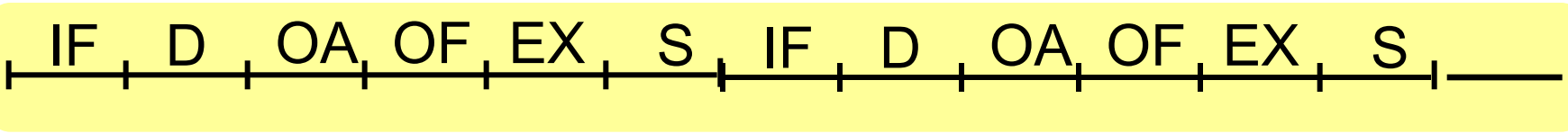
команд – Instruction Level Parallelism (ILP).
There are two largely separable approaches to exploiting of ILP.
Dynamic scheduling: hardware-intensive approaches, dominates in superscalar processors.
Static scheduling: compiler-intensive approaches, dominate in VLIW-processors (Itanium, Transmeta Crusoe).
Процесс выполнения команд состоит из k последовательных стадий (например: IF, D, OA, OF,EX, S).

First pipelined processor – IBM Stretch (1962).
Potential overlap among instructions is called instruction-level parallelism (ILP).
Потенциальное совмещение этапов выполнения различных команд называется параллелизмом уровня команд – Instruction Level Parallelism (ILP).

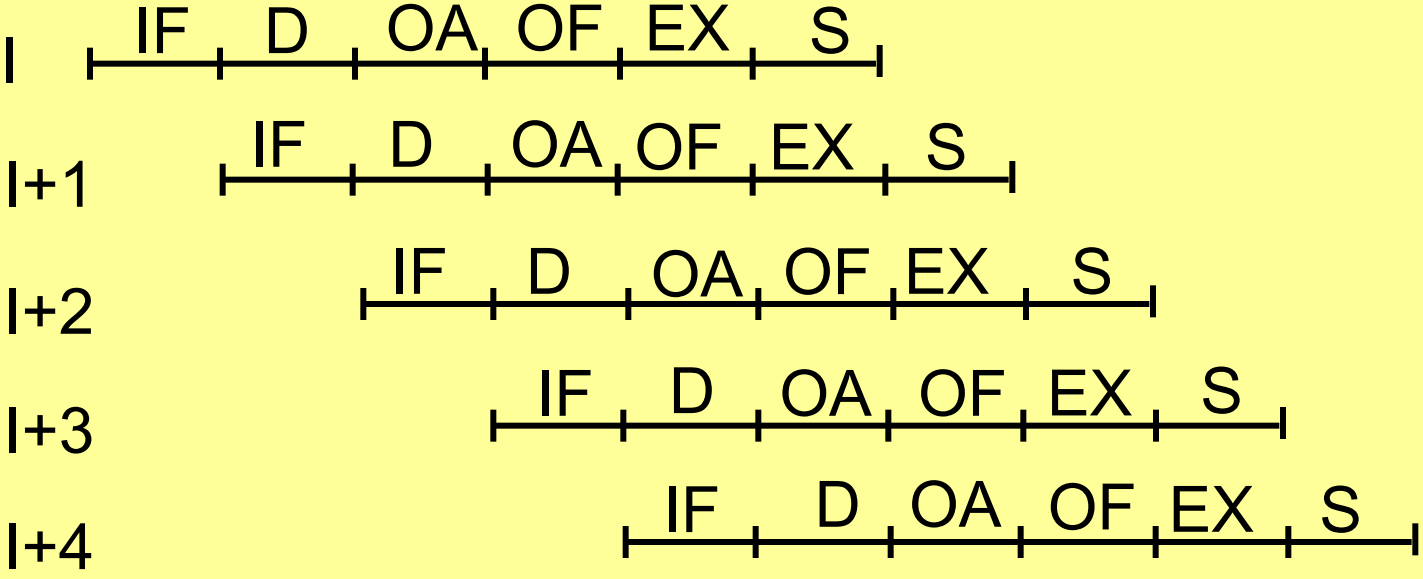
There are two largely separable approaches to exploiting of ILP.
Dynamic scheduling: hardware-intensive approaches, dominates in superscalar processors.
Static scheduling: compiler-intensive approaches, dominate in VLIW-processors (Itanium, Transmeta Crusoe).
Процесс выполнения команд состоит из k последовательных стадий (например: IF, D, OA, OF,EX, S).

IF

Unpipelined:



Pipelining allows to overlap different instructions at different stages.



Определение эффективности конвейеров.

Для характеристики эффективности используют три метрики: ускорение, эффективность и производительность.

Ускорение S – отношение времени обработки без конвейера к времени обработки с конвейером.

Наилучшее время $T_{КОН}$ обработки N команд входного потока на конвейере из K стадий и тактовым периодом **т** можно определить следующим способом:

$T_{КОН} = (K + (N-1))$ **т.**

В процессоре без конвейера общее время выполнения $T_{БЕЗ\ КОН} = NK$**т.**

$S = NK/(K + (N-1))$.

При $N \rightarrow \infty$ ускорение $S \rightarrow K$.

Метрики конвейера

Эффективность E – доля ускорения, приходящаяся на одну ступень конвейера.

$E = S/K = N/(K+(N-1))$.

Третьей метрикой является ***пропускная способность*** или производительность P .

$P = E/\tau = N/(K+(N-1)\tau)$.

При $N \to \infty$ эффективность стремится к 1, а производительность – к частоте тактирования.

Проблемы, возникающие на конвейере

CPI – Cycles Per Instructions

Pipeline CPI = Ideal pipeline CPI + Structural stalls +
+ Data hazards stalls + Control hazards stalls

By reducing each of the term of the right-hand side, we can minimize the overall pipeline CPI and thus increase IPC (Instruction Per Clock).

Структурные конфликты возникают из-за недостатка ресурсов: например, операционных устройств, при одновременном обращении в ОП за командами и данными и др.

Конфликты по данным

RAW (Read After Write).

Рассмотрим команды i и j (i предшествует j).

j пытается прочитать операнд-источник раньше, чем i его запишет.

WAR (Write After Read).

j пытается записать результат в приемник прежде, чем i его прочитает.

WAW (Write After Write).

j пытается записать операнд прежде, чем он будет записан командой i.

Устранение конфликтов по данным

.Программные методы

Оптимизирующий компилятор пытается создать такой код, чтобы между командами, которые могут конфликтовать друг с другом, находилось достаточное количество нейтральных команд.

В простейшем алгоритме компилятор планирует распределение команд в одном и том же базовом блоке.

Базовый блок – это линейный участок последовательности программного кода, в котором отсутствуют команды перехода – за исключением начала и конца блока (переходы внутрь этого участка также должны отсутствовать). Компилятор строит граф зависимости этих команд (потока операндов) и упорядочивает их таким образом, чтобы приостановок было меньше.

Для простых конвейеров стратегия планирования на основе базовых блоков вполне приемлема.

Однако для сложных конвейеров требуются более сложные алгоритмы.

Аппаратные методы

Фактическое разрешение конфликтов возлагается на аппаратные методы (суперскалярные процессоры).

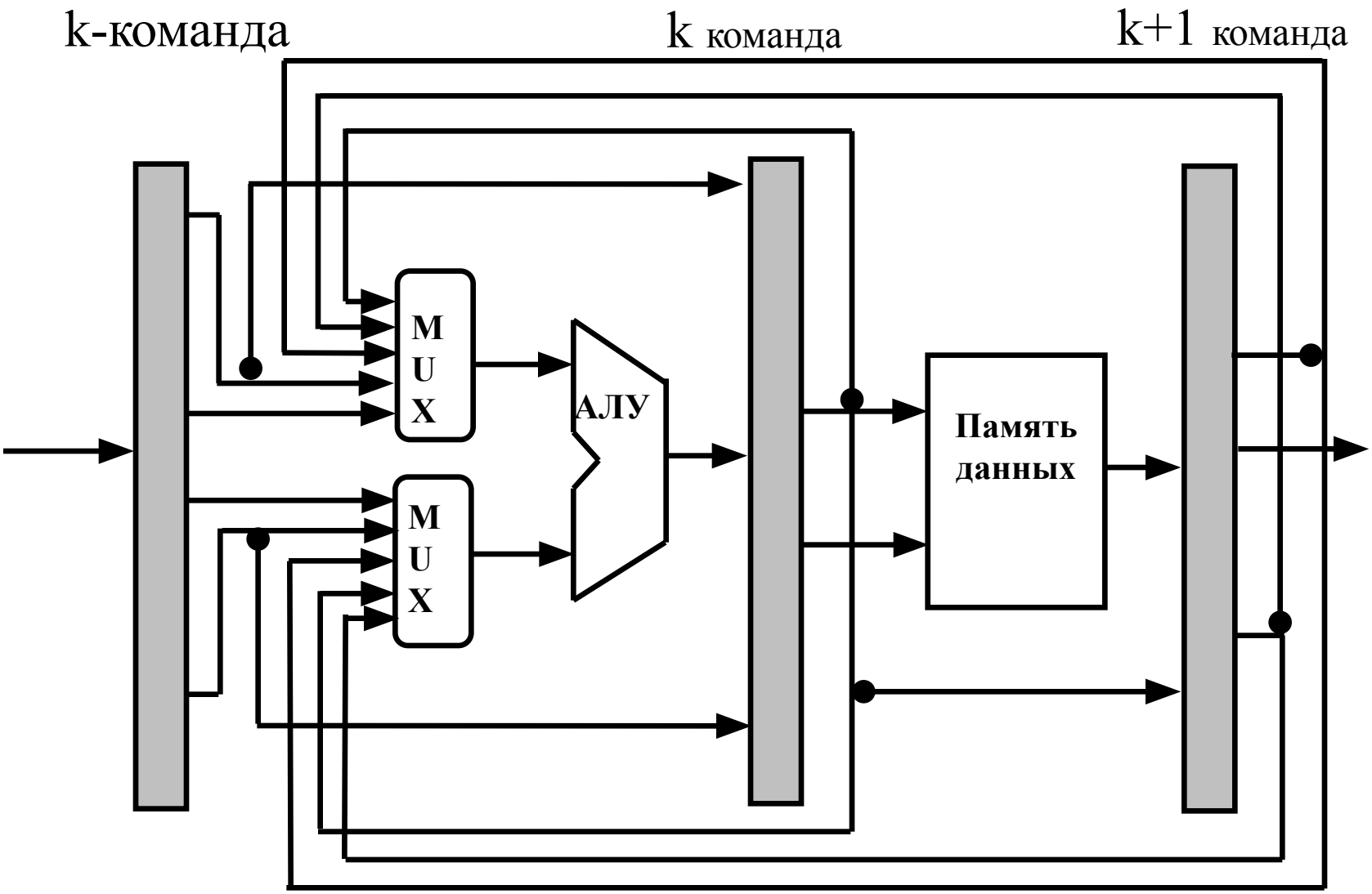
Очевидный метод – приостановка. Но он снижает производительность.

Применяется переименование регистров для конфликтов типа WAR и WAW.

Для конфликтов типа RAW используют прием ускоренного продвижения информации – data forwarding. Применяют тракты опережения и тракты обхода со средствами мультиплексирования.

АЛУ с цепями обхода (data bypassing) и ускоренной пересылки (data forwarding) показано ниже.

Bypassing



Переименование регистров (Registers Renaming)

Существенной зависимостью является только RAW. Зависимости WAR и WAW являются несущественными. Причины появления несущественных зависимостей: неоптимизированный программный код, ограниченное количество регистров, стремление к экономии памяти. Пример:

```
L2:
move    r3,  r7
lw      r8, (r3)
add     r3,  r3,4
lw      r9, (r3)
blez    r8,  r9,L3
```

Переименование регистров

Для преодоления несущественных WAR и WAW зависимостей используется механизм динамического отображения определяемых текстом программы логических ресурсов (ячеек памяти, регистров) на файл внутренних регистров. При этом подходе с одним логическим регистром может быть связано несколько значений в различных физических регистрах.

Число физических регистров обычно больше, чем логических. Физические регистры используются для временного хранения результатов до момента разрешения конфликтов по данным, после чего значение из внутреннего физического регистра переписывается в логический регистр.

K1: mov eax, 17

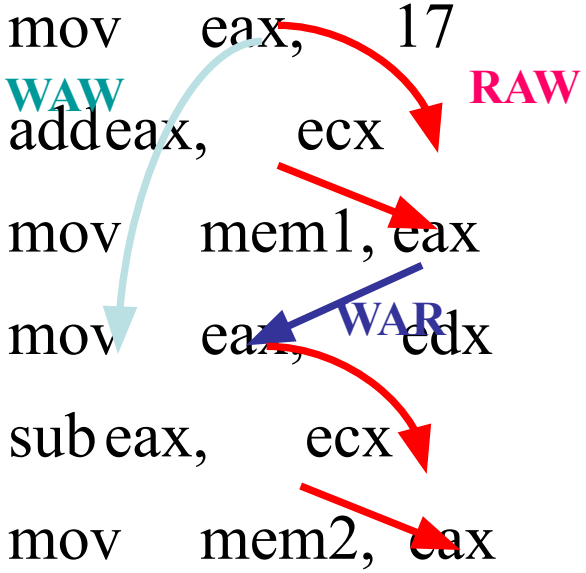
K2: add eax, ecx

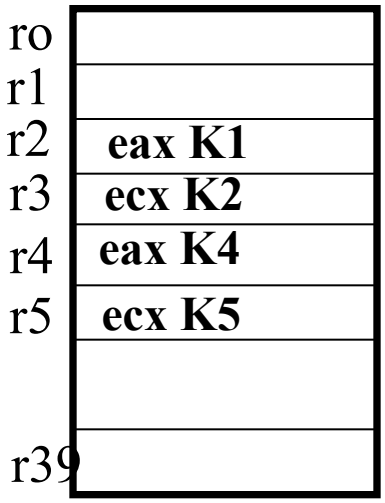
K3: mov mem1, eax

K4: mov eax, edx

K5: sub eax, ecx

K6: mov mem2, eax





После переименования

K1: mov r2, 17

K2: add r2, r3

K3: mov mem1, r2

K4: mov r4, r5

K5: sub r4, r3

K6: mov mem2, r4

Можно совместить выполнение

K1, K4

K2, K5

K3, K6

Пример конвейера

RISC – processor.

Basic instructions:

LOAD

- Instruction fetch - “IF” stage

1. PC → Mem, “Read”
2. Memory[PC] → IR
3. PC +4→PC
   - Decoding “D” stage
   - Operands Addressing “OA”stage

   (rb)+disp → addr
   - addr → Mem, “Read” “OF (EX)” stage

- Mem[addr] → rd “Write Back”

Пример конвейера

STORE

- Instruction fetch “IF” stage

1. PC → Mem, “Read”
2. Memory[PC] → IR
3. PC +4→PC
   - Decoding “D” stage
   - Operands Addressing “OA” stage

   (rb)+disp → addr
   - (rs) → Mem [addr], “Write” “S (EX)” stage

Пример конвейера

ADD/SUB

- Instruction fetch “IF” stage

1. PC → Mem, “Read”
2. Memory[PC] → IR
3. PC +4→PC
   - Decoding “D” stage
   - (rs) , (rt) → ALU, “ADD” “EX” stage
   - Result → rd “WB”

Memory[PC] → IR
PC +4→PC
Decoding
“EX” stage
(PC) + disp → Branch address
If Branch is taken then Branch address→ PC,

BR.Z

Instruction fetch

“IF” stage

PC → Mem, “Read”

Memory[PC] → IR

PC +4→PC

Decoding

“EX” stage

(PC) + disp → Branch address

If Branch is taken then Branch address→ PC,

PC → Mem, “Read”

Memory[PC] → IR

PC +4→PC

Decoding

(PC) + disp → Branch address “EX” stage

Branch address→ PC

“Load” instruction is executed for 5 cycles.

JMP

“IF” stage

Instruction fetch

PC → Mem, “Read”

Memory[PC] → IR

PC +4→PC

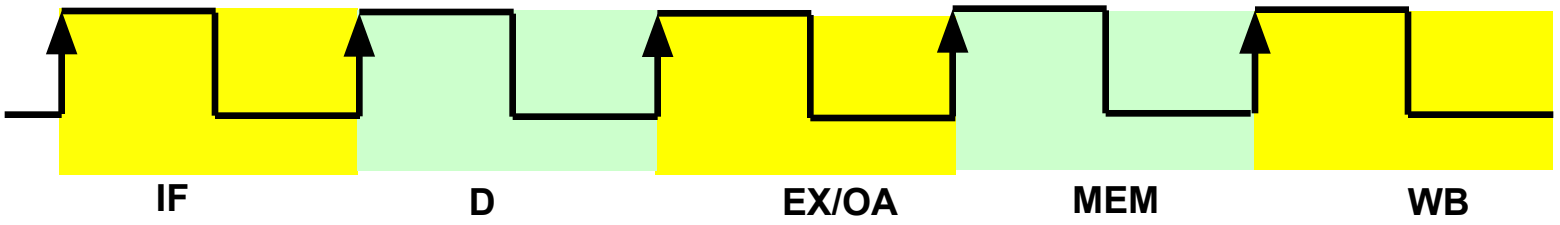
Decoding

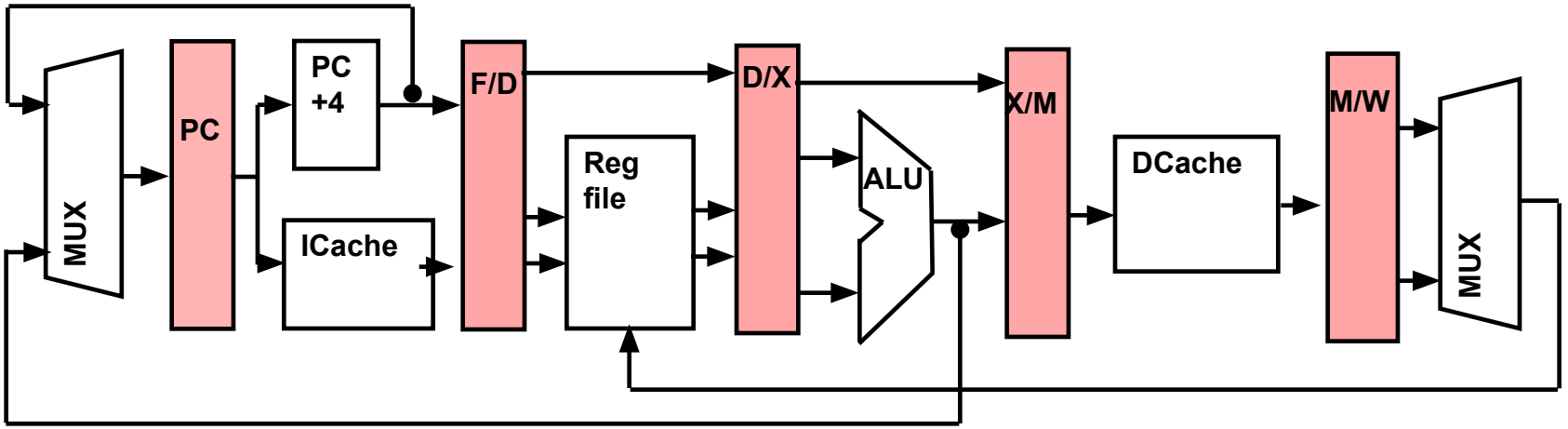
(PC) + disp → Branch address “EX” stage

Branch address→ PC

“Load” instruction is executed for 5 cycles.

Пример конвейера





Пример конвейера

Stages divided by *pipeline registers/latches.*

Pipeline registers

- Reg. PC contains PC
- Reg. F/D contains PC, undecoded instruction
- Reg. D/X contains PC, opcode, regfile[rs1], regfile[rs2], rd
- Reg. X/M contains opcode, regfile[rs], ALUOUT (result), rd
- Reg M/W contains ALUOUT (result), MEMOUT (ValM), rd

Data Hazards

Pipelines Limitations

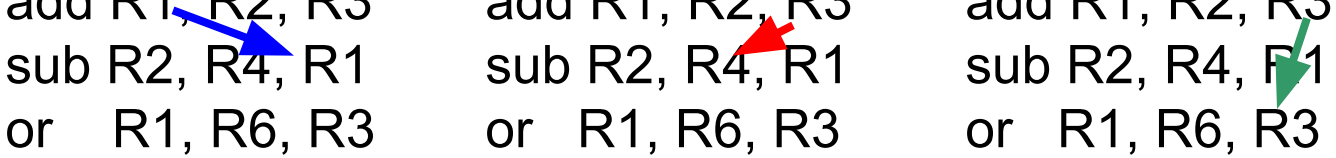
Structural hazards

Data hazards

Control hazards

Data Hazards

Two different instructions use the same storage location.



read-after-write (RAW) true dependence (real)

write-after-read (WAR) anti-dependence (artificial)

write-after-write (WAW) output dependence (artificial)

Конфликты по управлению

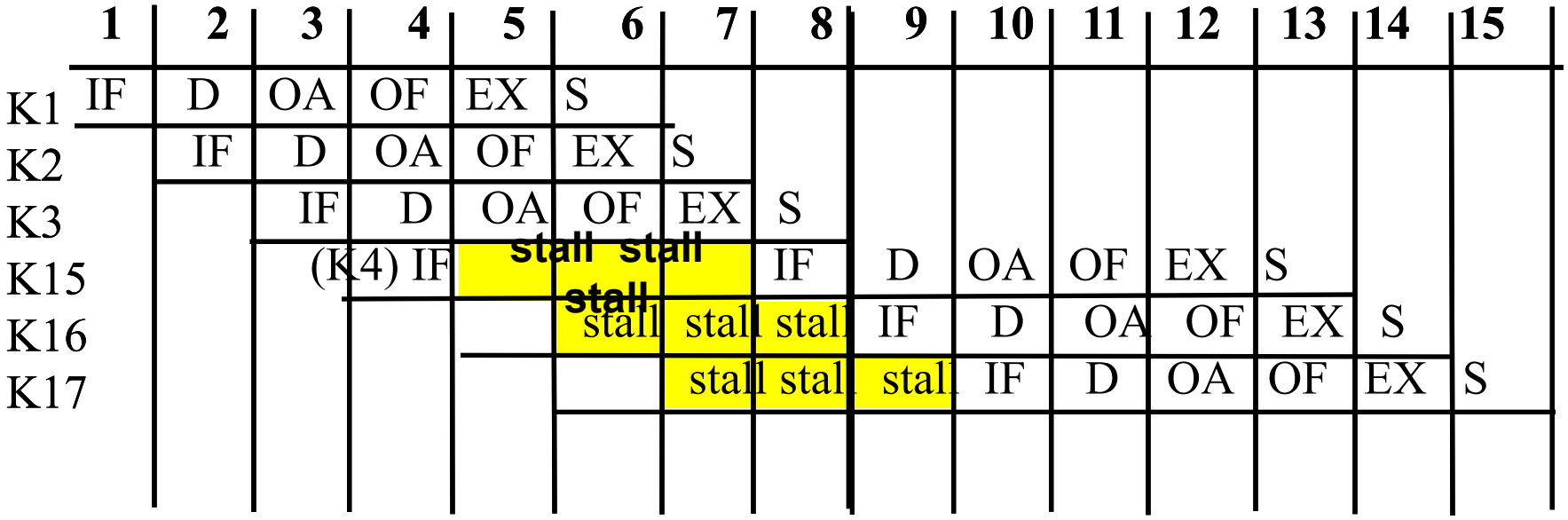
Методы решения проблемы условного перехода

Наибольшие проблемы при создании эффективного конвейера обусловлены командами, изменяющими последовательность вычислений: условный и безусловный переход, вызов процедуры, возврат из процедуры.

Используется несколько подходов при решении проблемы условного перехода.

Самый простой способ **– метод выжидания.**

В этом случае происходит подавление операций на конвейере после того, как будет обнаружена команда условного перехода.



На стадии декодирования команды К3 обнаружено, что К3 - команда условного перехода. После этого команда К4 (уже выбранная) снимается с конвейера. Конвейер приостанавливается (stalls). Новая команда будет загружена на конвейер только после того, как К3 пройдет стадию выполнения EX. Предположим, что происходит переход на команду К15. Тогда на стадии 8 произойдет ее выборка и конвейер будет загружен следующими после нее командами.

Если конвейер будет приостановлен на три такта на каждой команде условного перехода, то это существенно снижает производительность процессора. При частоте команд условного перехода, равной 30% и идеальном CPI, равном единице, процессор с приостановками условных переходов достигает примерно только половины ускорения, получаемого за счет конвейеризации команд.

Снижение потерь от условных переходов является критическим вопросом.

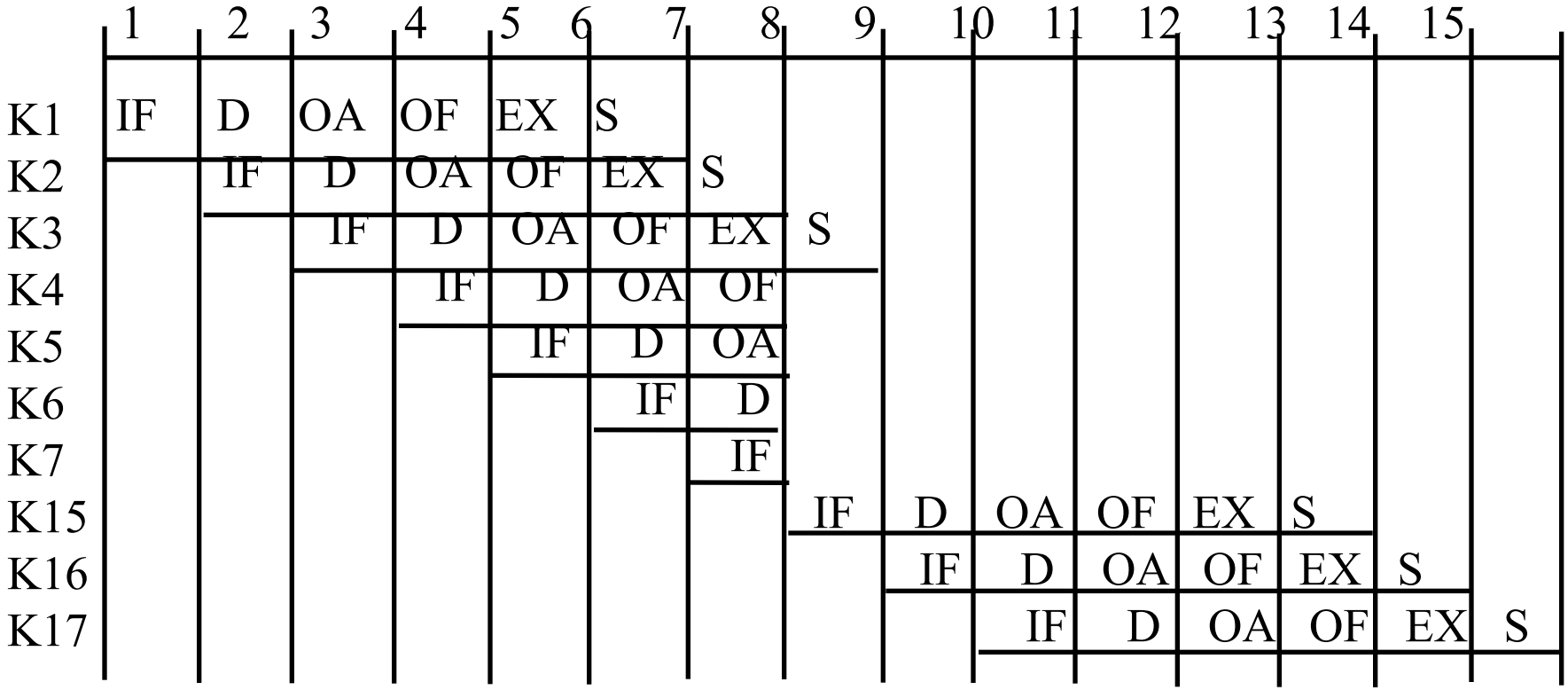
Число тактов, теряемых при приостановках из-за условных переходов может быть снижено двумя способами:

1. Обнаружением, является ли переход выполняемым или невыполняемым на более ранних стадиях конвейера.

2. Более ранним вычислением значения адреса перехода.

Метод возврата

Условный переход прогнозируется как невыполняемый. Выполнение программы на конвейере продолжается, как если бы переход вовсе не выполнялся.



Штраф за нарушение естественного порядка выполнения команд

Рассмотрим, что происходит, когда на конвейер попадает команда условного перехода. Предположим, что команда К3 оказалась командой условного перехода, передающей управление команде К15. До тех пор, пока команда К3 не пройдет стадию ЕХ, никакой информацией о том, будет переход или нет, устройство управления конвейером не располагает.

Предполагается, что переход не происходит. Поэтому после К3 на конвейер подается команда К4, К5, К6 и К7. В случае, показанном на рисунке, после прохождения командой К3 стадии ЕХ (на 7-м такте) выяснилось, что будет переход на К15.

На 8-м такте в конвейер была загружена команда К15 . На интервале от 9-го до12-го такта ни одна команда с конвейера не сошла. Это и есть “штраф” за изменение естественной последовательности команд.

Этот метод использован в VAX/11.

Предсказание переходов

Предсказание переходов является одним из наиболее эффективных способов борьбы с конфликтами по управлению.

Идея заключается в том, что еще до момента выполнения команды условного перехода или же сразу после ее поступления делается предположение о наиболее вероятном результате выполнения такой команды (переход произойдет или нет). Последующие команды подаются на конвейер в соответствии с предсказанием.

При ошибочном предсказании конвейер необходимо вернуть к состоянию, с которого началась выборка ненужных команд, т.е. очистить конвейер, что эквивалентно приостановке конвейера. Цена ошибки может быть достаточно высокой, но при правильном предсказании имеем большой выигрыш – конвейер функционирует ритмично, без остановок. Выигрыш тем больше, чем выше точность предсказания.

Предсказание переходов (Prediction)

Точность предсказания – процентное отношение числа правильных предсказаний к их общему количеству.

Доказано, что для того, чтобы снижение производительности конвейера из-за приостановок по управлению не превысило 10%, точность предсказания должна быть выше 97,7%.

В зависимости от того, когда и на основе какой информации делается предсказание, используют два подхода: статический и динамический.

Статическое предсказание

Статическое предсказание делается на этапе компиляции программы.

Используют следующие стратегии:

- Переход происходит всегда (ПВ);
- Переход не происходит никогда (ПН);
- Предсказание определяется по результатам профилирования;
- Предсказание определяется кодом операции команды перехода;
- Предсказание зависит от направления перехода;
- При первом выполнении команды переход имеет место всегда.

При предсказании *на основе кода операции* предполагается, что для одних команд переход произойдет, а для других – нет.

Стратегия ПВ – для команд перехода по условию <0,>=0,=0, а ПН – всем остальным командам. Стратегия приводит к успеху более, чем в 75% (по Смиту – 86%).

Эффективность предсказания зависит от характера программы.

Исходя из *направления перехода* – переходу “назад” назначается стратегия ПВ, а переходу “вперед” – ПН.

Для перехода “назад” фактический переход имеет место в 85% случаев, а для перехода “вперед” – 65%.

В последней стратегии *при первом выполнении команды перехода предсказывается, что переход обязательно будет.* Точность выше, чем у всех предшествующих. Но этот метод практически нереализуем при больших объемах программ.

Branch Prediction

Ветвления бывают случайные и регулярные.

Регулярные ветвления довольно хорошо предсказываются на основе предыдущей статистики их выполнения.

Случайные ветвления в принципе невозможно предсказать на основании сбора предыдущей статистики их выполнения.

Динамическое предсказание

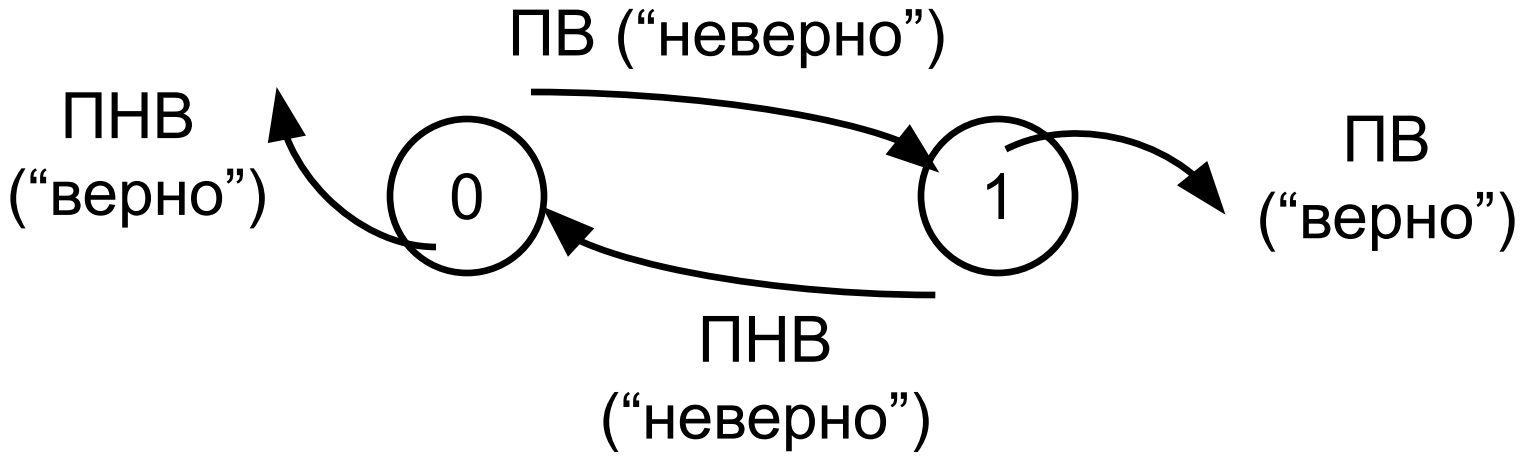
Идея динамического предсказания ветвлений предполагает накопление информации о предшествующих переходах. Решение о наиболее вероятном исходе команды условного перехода принимается, исходя из этой информации.

Динамические стратегии по сравнению со статическими, обеспечивают более высокую точность предсказания.

Динамические предсказатели ветвления производят анализ предыстории выполнения перехода. Каждой команде перехода ставится в соответствие один или несколько битов прогноза.

В самом простом случае рассматривается результат самого последнего выполнения команды (однобитовая схема предсказания ветвлений). Бит предсказания ветвлений в однобитовой схеме называют переключателем – “принято/не принято” (taken /not taken).

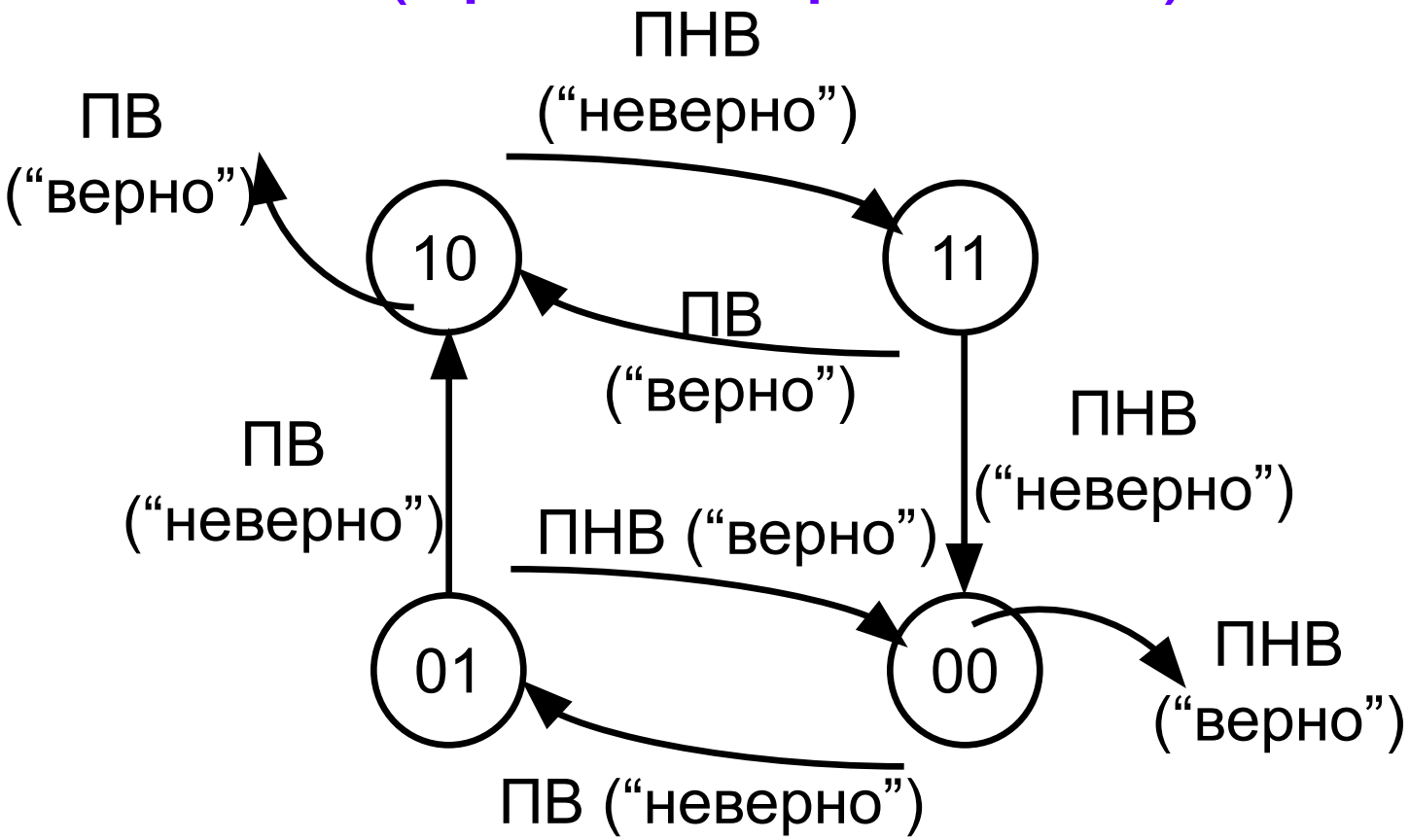
1-битная схема прогноза



Состояние “0” – предсказывается, что перехода не будет;
Состояние “1” – предсказывается, что переход будет.
Сигналы ПВ (переход выполняется) и ПНВ (переход не выполняется) поступают на вход автомата из блока выполнения команды ветвления (Branch unit) – обратная связь. С их помощью осуществляется коррекция прогноза.

Точность прогноза недостаточная – направление перехода в цикле дважды будет предсказываться неверно.

Двухбитная схема прогноза (предиктор Смита)



00 – очень малая вероятность перехода (сильное предсказание)
01 – малая вероятность перехода (слабое предсказание)
10 – очень большая вероятность перехода (сильное предсказание)
11 – большая вероятность перехода (слабое предсказание)

Когда автомат находится в состоянии 00, то предсказывается, что перехода не будет not taken).

Если же в действительности переход будет выполнен, то автомат перейдет в состояние 01. Это означает, что когда данная команда перехода встретится в очередной раз, блок выборки команд снова предскажет невыполнение перехода. Если и в этот раз переход был предсказан неверно, т.е. произошел, то автомат перейдет в состояние 10. После этого будет предсказываться выполнение перехода (taken). Таким образом, прежде, чем предсказатель изменит значение предсказания, он должен подряд два раза дать неверное предсказание.

Для выбора правильного состояния автомата при входе в цикл можно использовать статический прогноз с установкой начального состояния автомата 11. В этом случае предсказание всегда будет верным, за исключением последней итерации цикла.

Последнего неверного предсказания не избежать. Точность предсказания для рассмотренного выше цикла с 10-ю итерациями будет 90%.

Двухуровневые схемы предсказаний

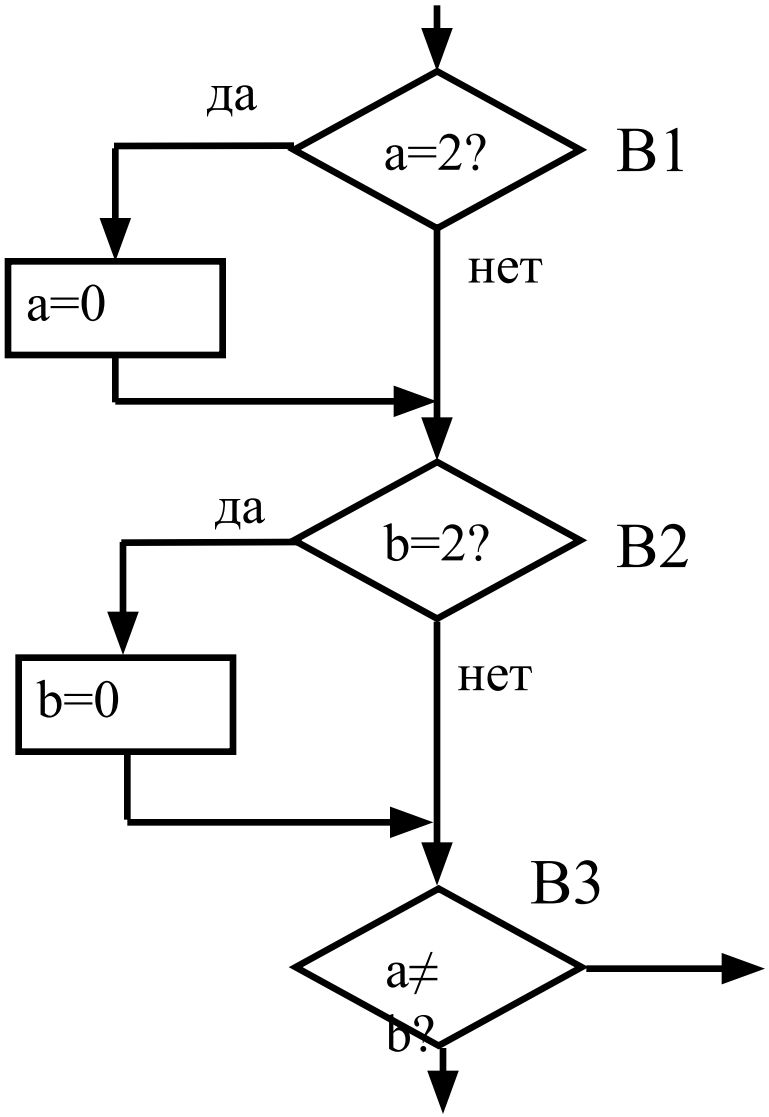
Одноуровневые схемы предсказания ориентированы на те команды условного перехода, исход которых существенно зависит от их собственных предыдущих исходов.
В то же время для многих команд наблюдается сильная зависимость не от собственных исходов, а от результов выполнения других, предшествующих им команд ветвления.
Это обстоятельство учитывают двухуровневые адаптивные предикторы.
Их называют коррелированными, т.к. они отражают взаимозависимости между командами переходов.

Двухуровневые схемы предсказаний

Рассмотрим небольшой фрагмент из текста программы тестового пакета SPEC92 (наихудший случай для двухбитной схемы).

```
if (a= = 2)
a=0;
if (b= = 2)
b=0;
if (a !=b)  { …
```

Двухуровневые схемы предсказаний




```
     cmp   eax, 2
     jne   l1;  B1
     xor   eax, eax
l1:  cmp   ebx, 2
     jne   l2;  B2
     xor   ebx,ebx
l2:  cmp   eax,ebx
     je    l3;  B3
```

Если оба перехода B1 и B2 были выполняемыми, значит переход B3 будет невыполняемым. Одноуровневая схема прогнозирования никогда этого не учтет.

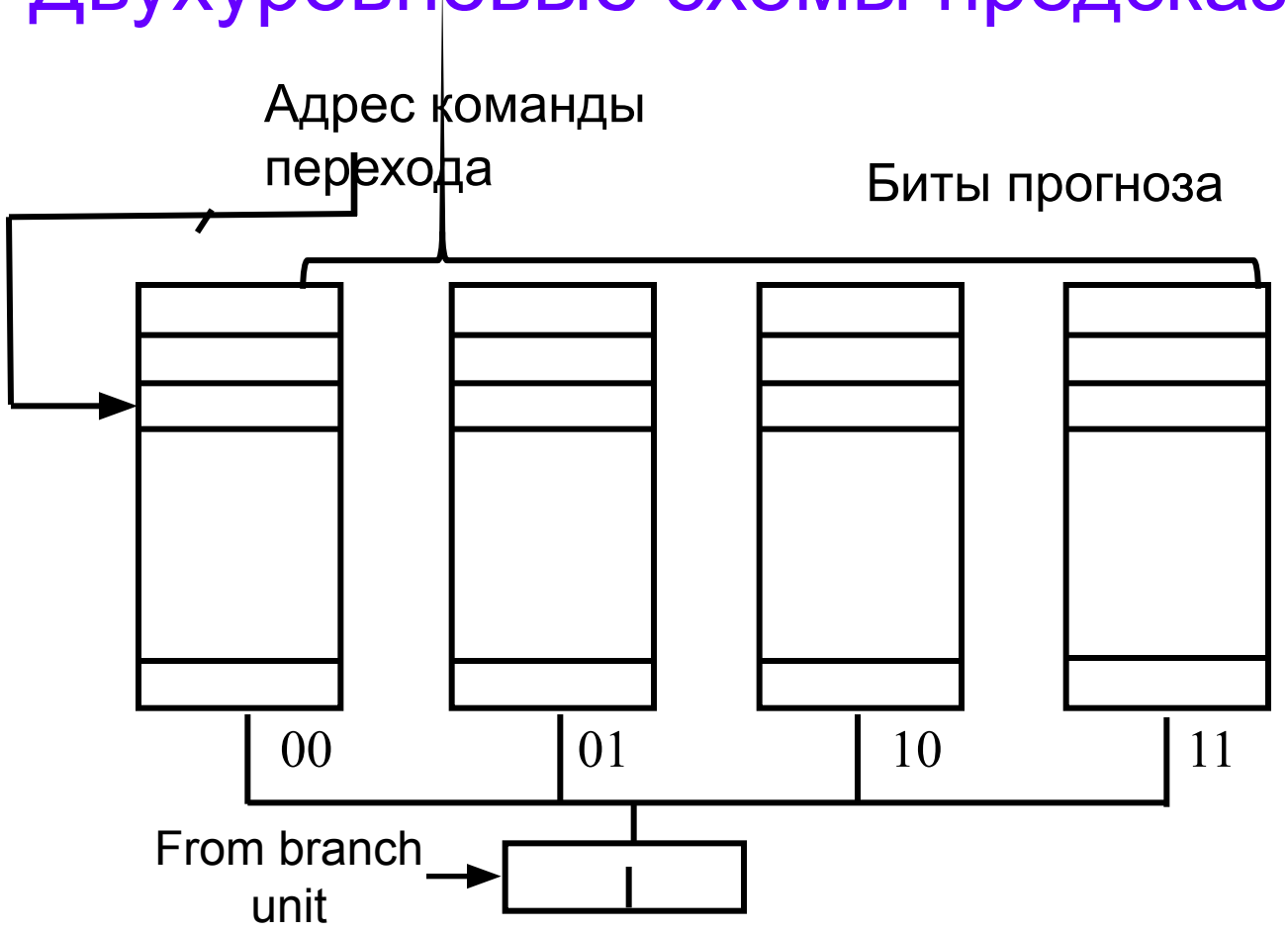
Двухуровневые схемы предсказаний

Можно проследить, были ли совершены k последних команд переходов, независимо от того, какие это были команды переходов. Для этого используется k-битное число, которое хранится в k-битном сдвигающем регистре глобальной предыстории. Каждой комбинации k битов глобальной предыстории соответствует таблица n битов прогноза.

Схема прогнозирования называется прогнозом (k,n), если она использует поведение последних k команд условного перехода для выбора одной из 2^k схем прогнозирования, каждая из которых представляет собой n-битную схему предсказания для каждого отдельного перехода.

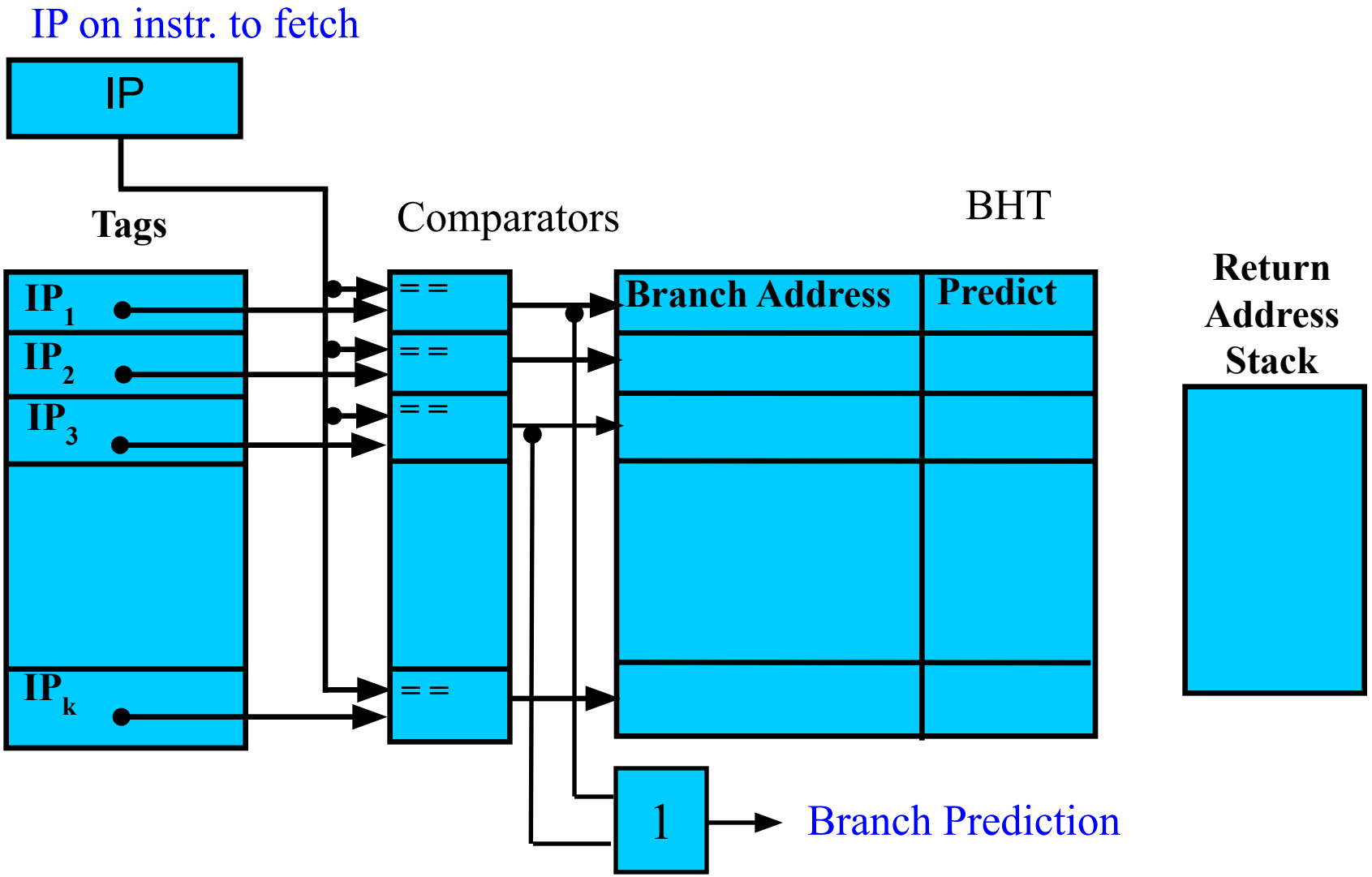
Схема двухуровневого предсказания дает более высокий процент успешного прогнозирования и использует небольшое количество дополнительной аппаратуры.

Двухуровневые схемы предсказаний

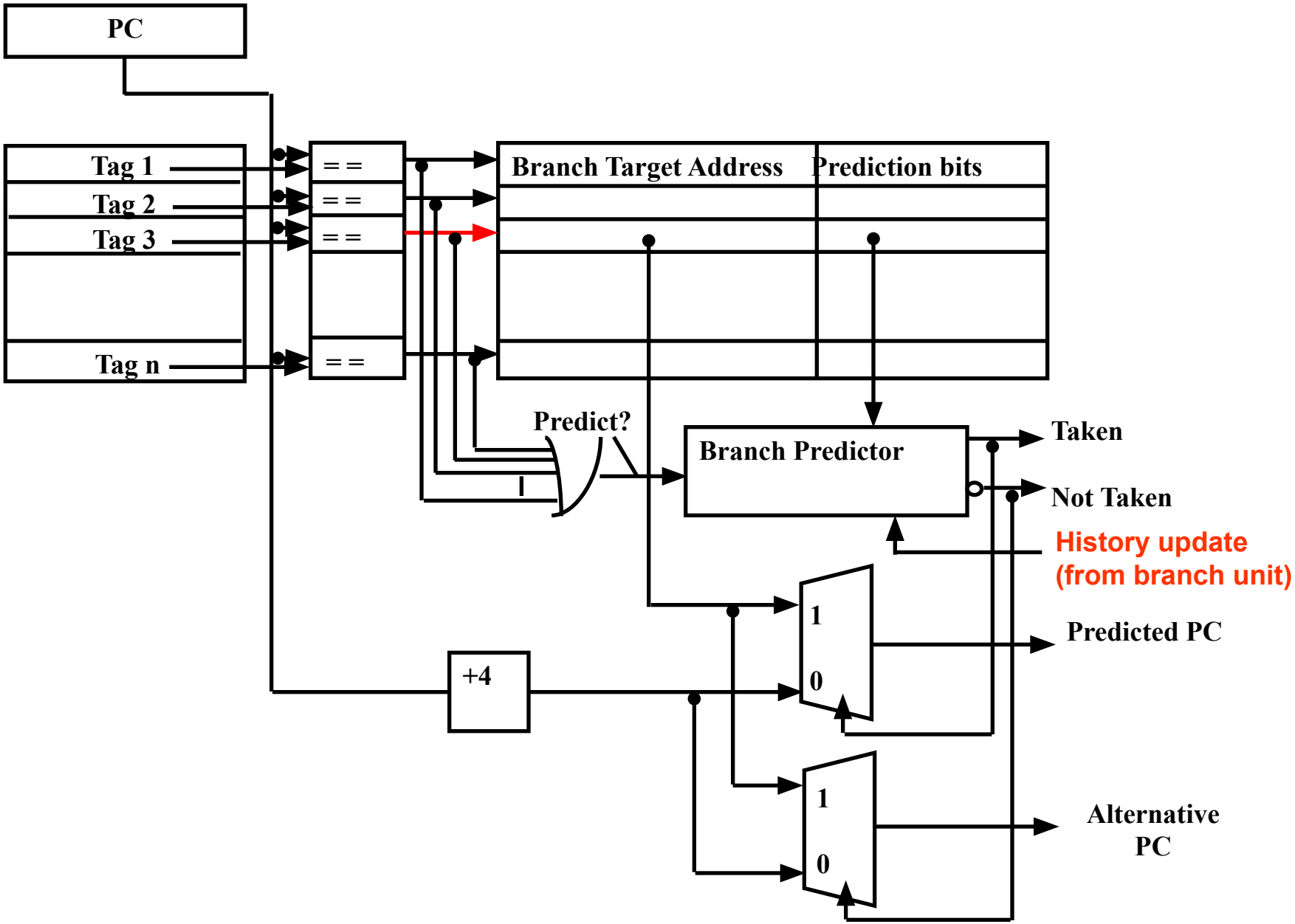


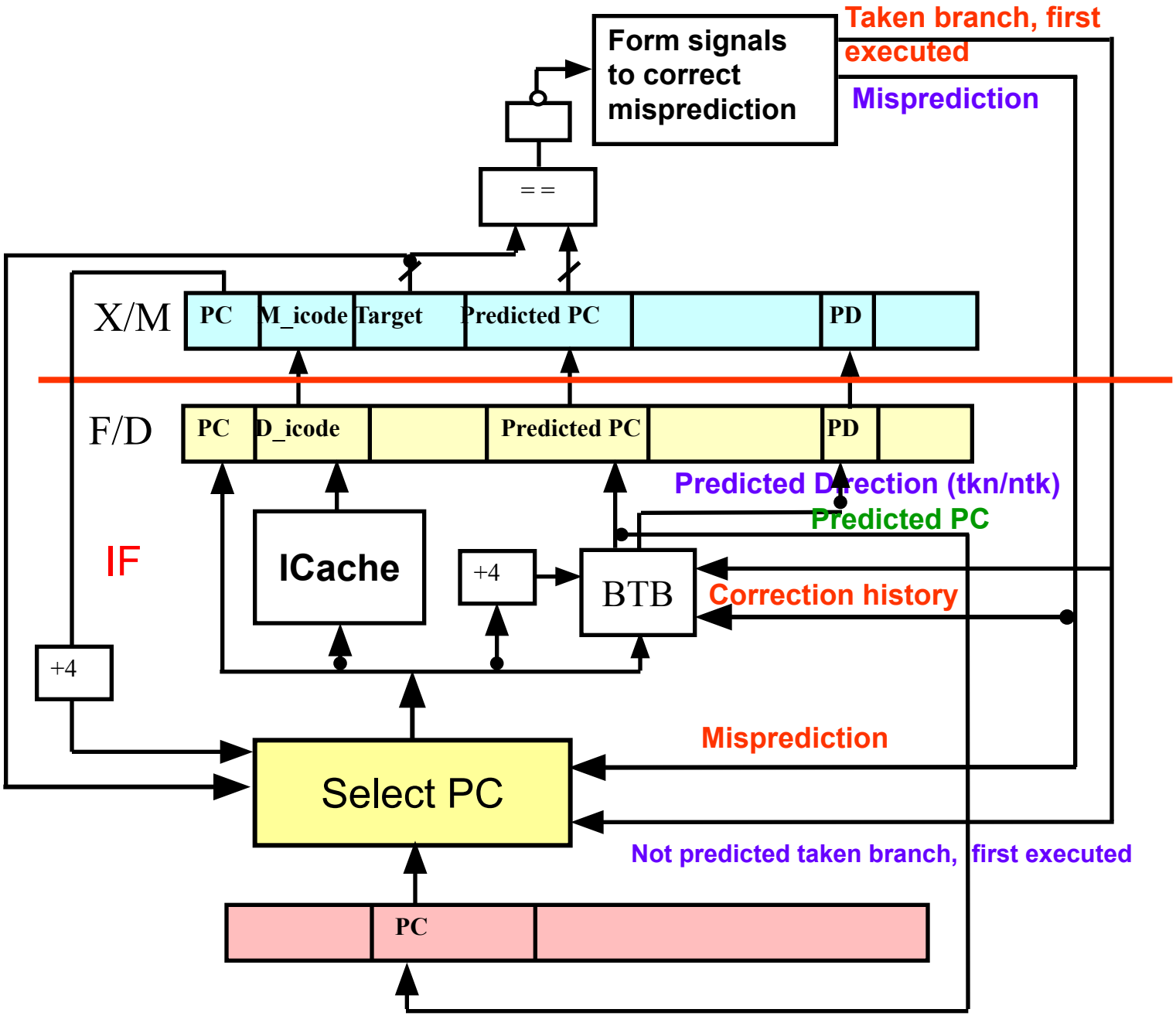
Представлена BHT предиктора (2,2) – 2бита глобальной истории и 2 бита локальной истории (например, по счетчику).
После выполнения команды содержимое счетчика одной ветви обновляется.

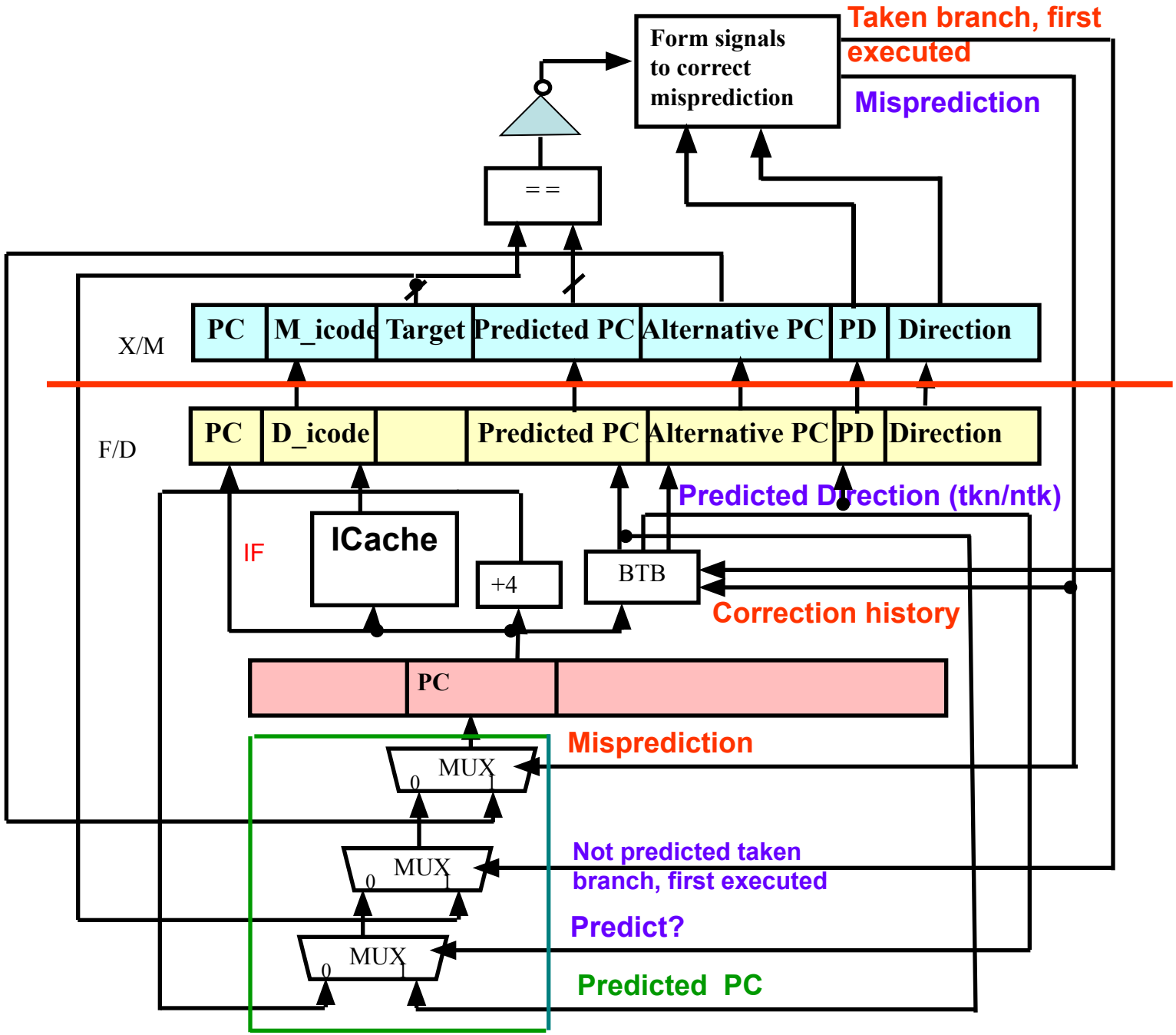
Branch Target Buffer



BTB является ассоциативным FIFO буфером.


PC
Tag 1
Tag 2
Tag 3
Tag n
= =
= =
= =
= =
Branch Target Address
Prediction bits
Predict?
Branch Predictor
Taken
Not Taken
History update
(from branch unit)
1
0
Predicted PC
+4
1
0
Alternative
PC


Form signals to correct misprediction
Taken branch, first executed
Misprediction
= =
X/M
PC
M_icode
Target
Predicted PC
PD
F/D
PC
D_icode
Predicted PC
PD
Predicted Direction (tkn/ntk)
Predicted PC
IF
ICache
+4
BTB
Correction history
+4
Misprediction
Select PC
Not predicted taken branch, first executed
PC


Form signals to correct misprediction
Taken branch, first executed
Misprediction
==
X/M
PC
M_icode
Target
Predicted PC
Alternative PC
PD
Direction
F/D
PC
D_icode
Predicted PC
Alternative PC
PD
Direction
Predicted Direction (tkn/ntk)
IF
ICache
+4
BTB
Correction history
PC
Misprediction
MUX
0
1
Not predicted taken branch, first executed
Predict?
Predicted PC

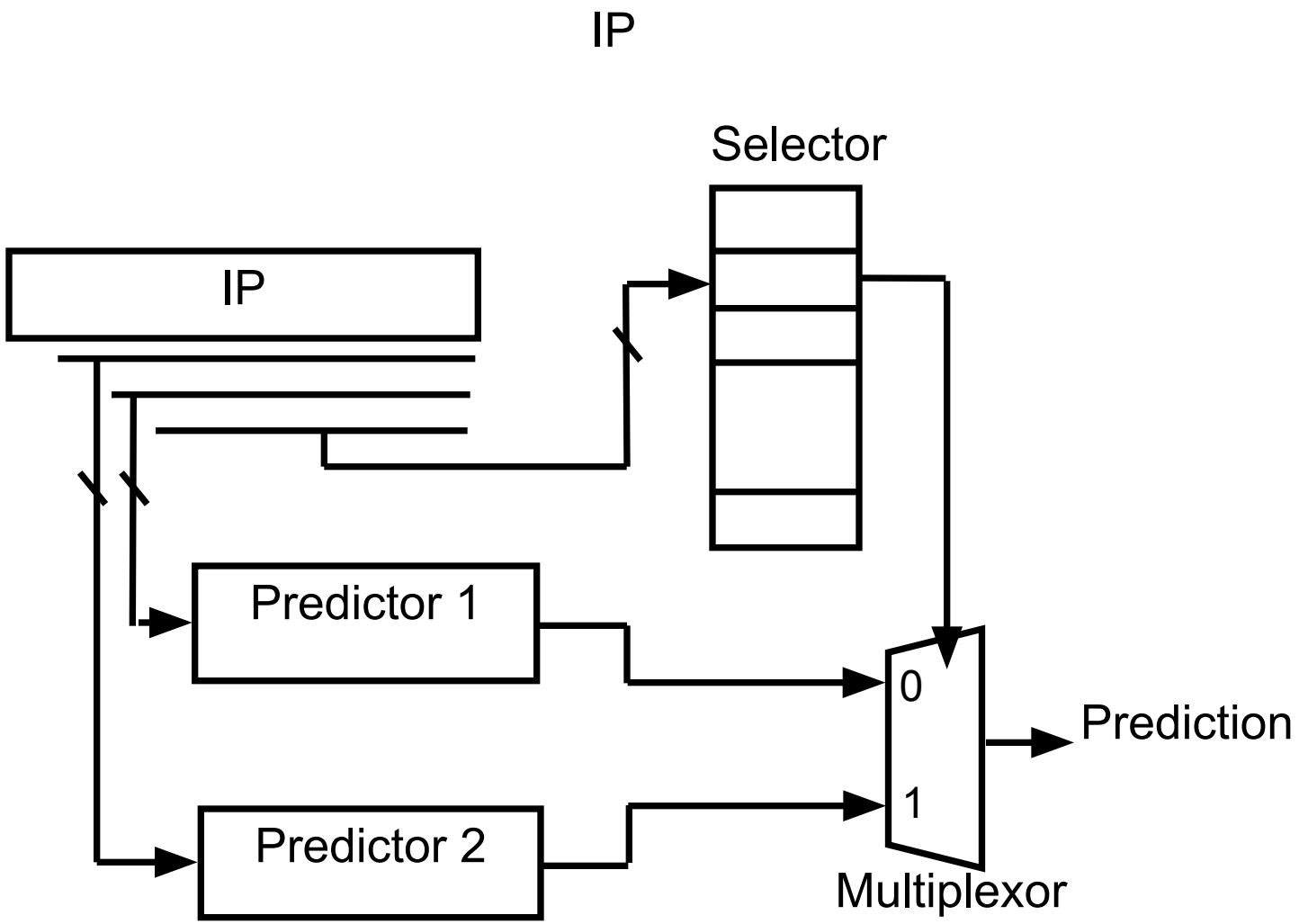
Гибридные схемы предсказания

Характерна сильная зависимость точности предсказания от особенностей программ, в которых эти стратегии реализуются.

Схема предсказания, прекрасно проявляя себя с одними программными продуктами, с другими может давать совершенно неудовлетворительные результаты.

Точность предсказания улучшается с увеличением глубины предыстории переходов, но это происходит лишь после накопления определенной информации (этот период принято называть временем “разогрева”).

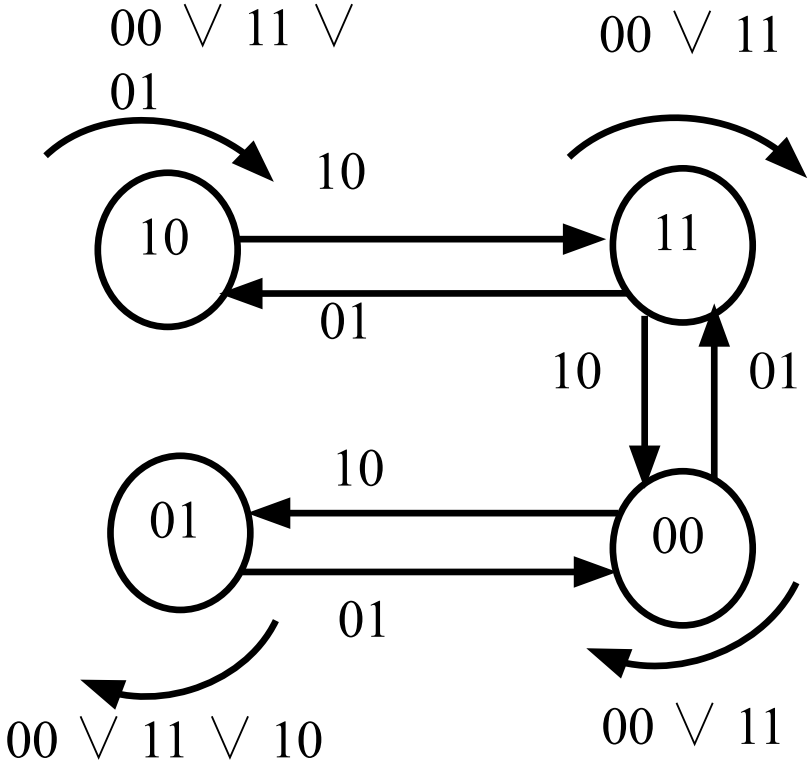
Чтобы уменьшить зависимость точности предсказания от особенностей программ, в которых эти стратегии реализуются, используют гибридные схемы предсказания.


IP
Selector
IP
Predictor 1
Predictor 2
0
1
Prediction
Multiplexor

Селектор

Обычно для выбора селектора используют 2-битный счетчик с насыщением.

Граф переходов автомата Мура, описывающего поведение селектора:



Селектор

00 – оба предиктора предсказали неправильно.
11 – оба предиктора предсказали правильно.
10 – предсказание первого верно, второго – неверно.
01 – предсказание первого предиктора неверно, второго – верно.

Выбор предиктора осуществляется старшим разрядом счетчика, который поступает на управляющий вход мультиплексора.

Если состояния 00 ∨ 01 – выбирается первый предиктор, если 10 ∨ 11 – выбирается второй предиктор.

По имеющимся оценкам, точность предсказания гибридных схем повышается по сравнению с бимодальными до 97%.