



Паралелно Програмиране

Съвременни паралелни архитектури.

Класификация на Флин.

CPU, GPGPU, HSA, HPC и др.

Класификация на Флин */Flynn/*

Класификация на Флин /Flynn/ (1966)

SISD

Старите Mainframe,
класическите PC и др.

SIMD

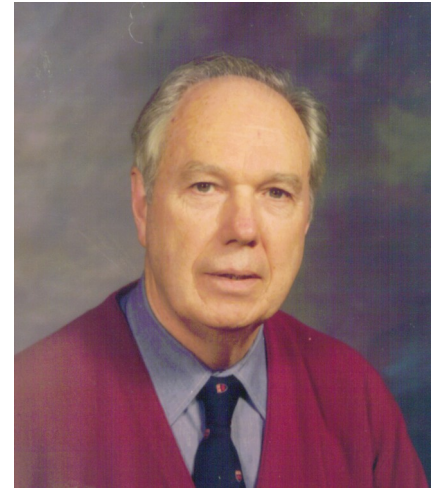
Повечето съвременни
PC, GPU и др.

MISD

Използва
за системи защитени
от повреди,
С.mmp computer и др.

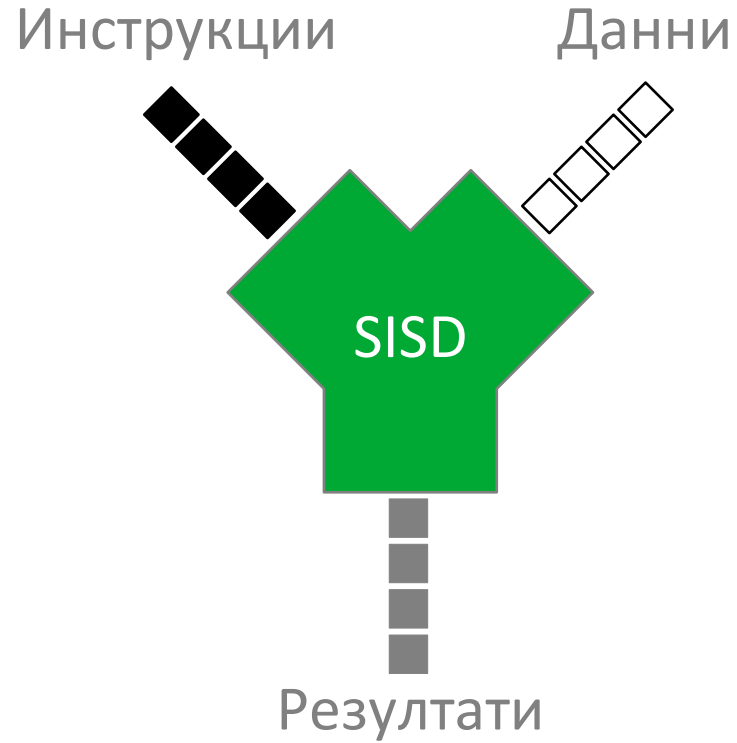
MIMD

Много-ядрените
суперскаларни проц.,
разпределените
системи и др.

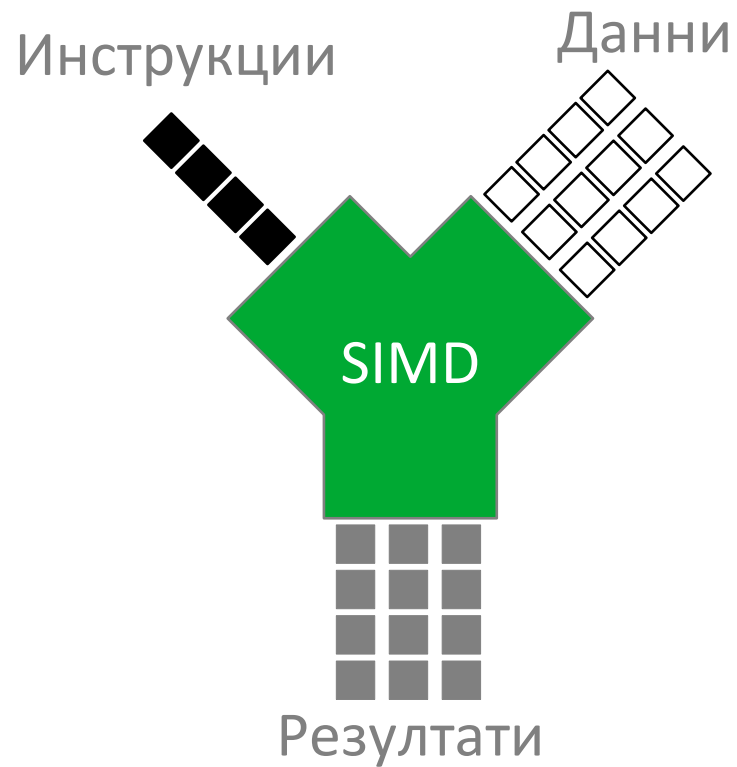


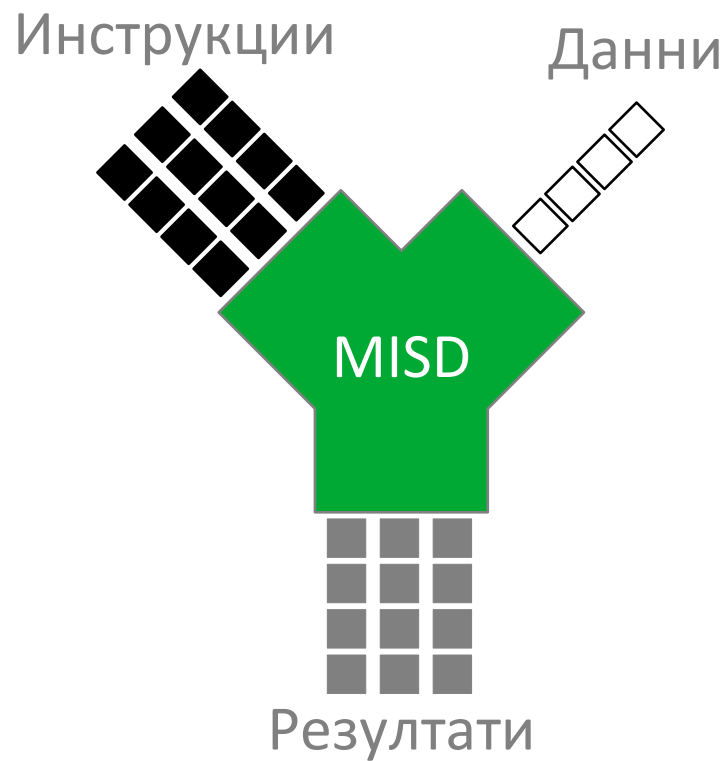
Класификация на Флин /Flynn/

- ❖ SISD – един поток инструкции над едни поток данни;
- ❖ SIMD – един поток инструкции над много потоци данни;
- ❖ MISD – много потоци инструкции над едни поток данни;
- ❖ MIMD – много потоци инструкции над много потоци данни;
 - ❖ SPMD – една програма, много потоци данни;
 - ❖ MPMD – много програми, много потоци данни;



SIMD

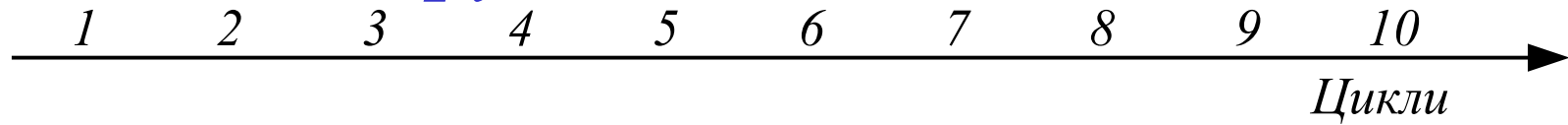






Класически Процесори

Изпълнение на инструкции

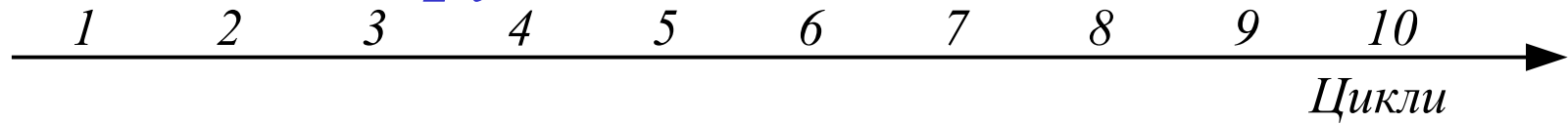


Инструкция №

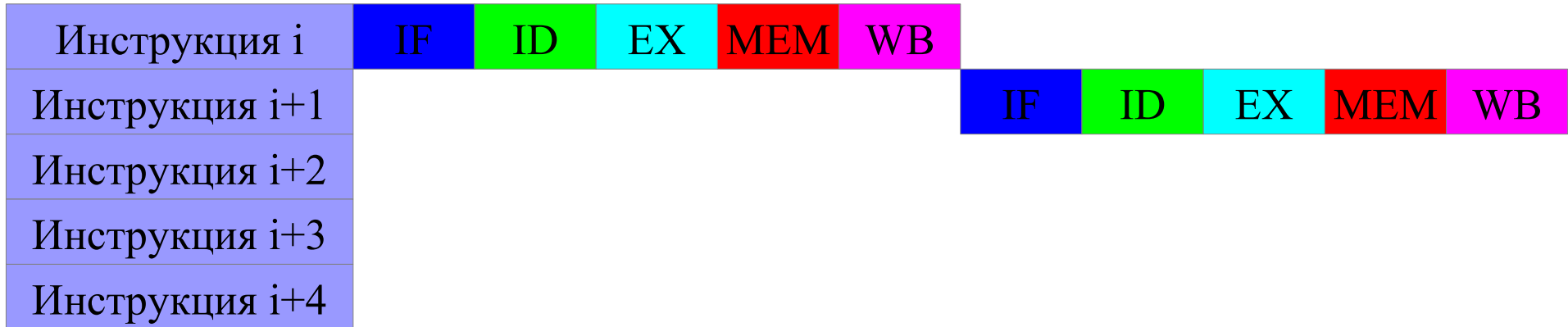


Последователно изпълнение на инструкциите.

Изпълнение на инструкции



Инструкция №

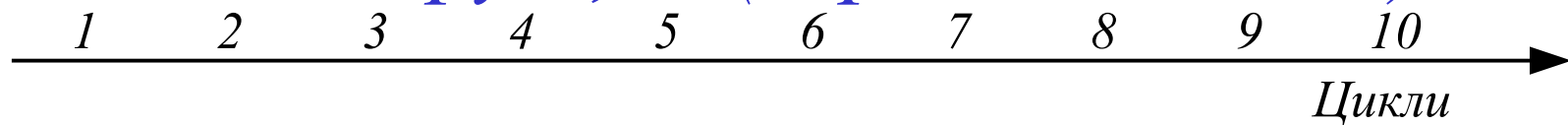


- ❖ IF – Instruction Fetch
- ❖ ID – Instruction Decode
- ❖ EX – Execution
- ❖ MEM – Memory Access
- ❖ WB – Write Back

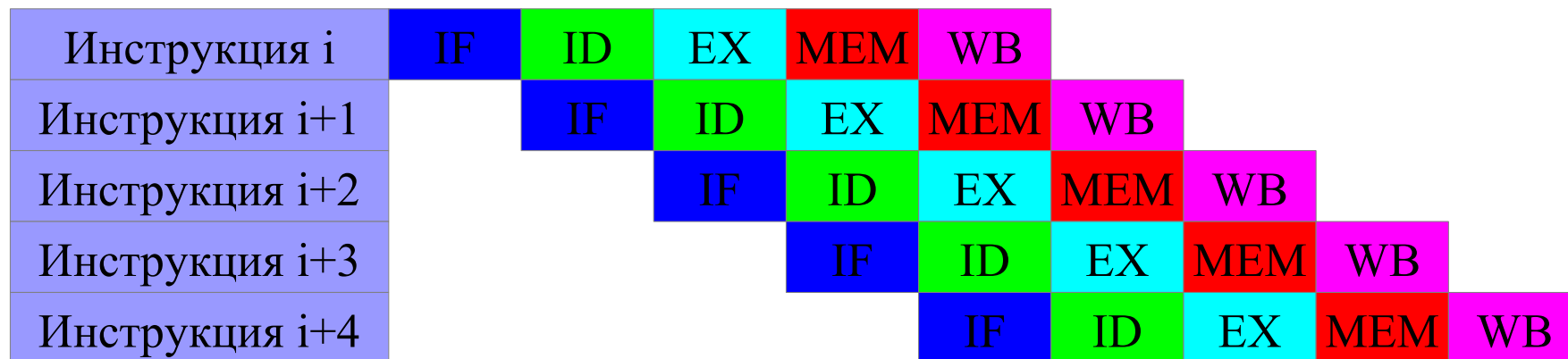
Но самите инструкции обикновено се състоят от отделни под етапи.

Скаларни/Pipelined Процесори

Изпълнение на инструкции (Pipelined/Scalar)



Инструкция №



- ❖ IF – Instruction Fetch
- ❖ ID – Instruction Decode
- ❖ EX – Execution
- ❖ MEM – Memory Access
- ❖ WB – Write Back

Характерно е и за класическите RISC архитектури.

Ограничения при конвейерното изпълнение

Опасностите (hazards) не позволяват следващата инструкция да бъде изпълнена по време на определения ѝ цикъл. Те се делят на:

- ❖ Даннови опасности (data hazards)

Инструкция зависи от резултата на друга инструкция, която е още в конвейера

- ❖ Контролни опасности (control hazards)

Причинени са от забавянето между извличането на инструкции и решенията за промени в потока инструкции (control flow) – условни, безусловни и индиректни преходи, подпрограми

- ❖ Структурни опасности (structural hazards)

Опити за използване на един и същ хардуер за две различни неща едновременно

Даннови опасности: без зависимост

Инструкция 3 не зависи от данните, получени от инструкция 2.
В двете инструкции се чете само от данните получени в инструкция 1.

```
1.  A = 3
2.  B = A
3.  C = A
```

Ако две инструкции не са взаимнозависими от данните си те обикновено могат да бъдат:

- ❖ Изпълнени едновременно;
- ❖ Напълно да се застъпват в конвейера;
- ❖ Да се изпълняват непоследователно (out-of-order);

При настъпване на такава ситуация в конвейера се казва, че имаме четене след четене (RAR), което не е риск

Даннови опасности: истинска зависимост

Инструкция 3 зависи от данните, получени от предходната инструкция 2

1. $A = 3$
2. $B = A$
3. $C = B$

Ако две инструкции са взаимнозависими от данните си те не могат да бъдат:

- ❖ Изпълнени едновременно;
- ❖ Напълно да се застъпват в конвейера;
- ❖ Да се изпълняват непоследователно (out-of-order);

При настъпване на такава ситуация в конвейера се казва, че има риск от четене след запис (RAW hazard)

Даннови опасности: анти зависимост

Инструкция 3 зависи от
инструкция 2 (инстр. 2 трябва да
се изпълни преди инстр. 3)

1. $B = 3$
2. $A = B + 1$
3. $B = 7$

Нарича се още зависимост от имената

Когато две инструкции използват един и същи регистър или адрес в паметта, но няма поток от данни между тези две инструкции. Има две версии на този тип зависимост

При настъпване на такава ситуация в конвейера се казва, че има риск от
запис след четене (WAR hazard)

Даннови опасности: изходна зависимост

Инструкция 3 зависи от
инструкция 1 (инстр. 1 трябва да
се изпълни преди инстр. 3)

1. **A** = 2 * X
2. B = A / 3
3. **A** = 9 * Y

Нарича се още зависимост от имената

Когато две инструкции използват един и същи регистър или адрес в паметта, но няма поток от данни между тези две инструкции. Има две версии на този тип зависимост

При настъпване на такава ситуация в конвейера се казва, че има риск от
запис след запис (WAW hazard)

Зависимост “от имената”

Зависимостта от имената може да се избегне като се смени името в инструкцията, така че да няма конфликт:

❖ Софтуерно преименуване (на регистри)

Прави се от компилатора

❖ Хардуерно преименуване (на регистри)

Прави се от процесора. Недостатък: по време на изпълнение

```
1. B = 3
2. A = B + 1
3. B = 7
```

```
1. B = 3
2. B2 = B
3. A = B2 + 1
4. B = 7
```

WAR опасност

```
1. A = 2 * X
2. B = A / 3
3. A = 9 * Y
```

```
1. A2 = 2 * X
2. B = A2 / 3
3. A = 9 * Y
```

WAW опасност

Контролни опасности

Всяка инструкция е зависима от някакво множество преходи. Тези зависимости трябва да се спазват, за да може да се запази семантиката на програмата

```
if c1 {  
    I1;  
}  
if c2 {  
    I2;  
}
```

I1 зависи от условието c1, а I2 зависи от условието c2, но не и от c1.

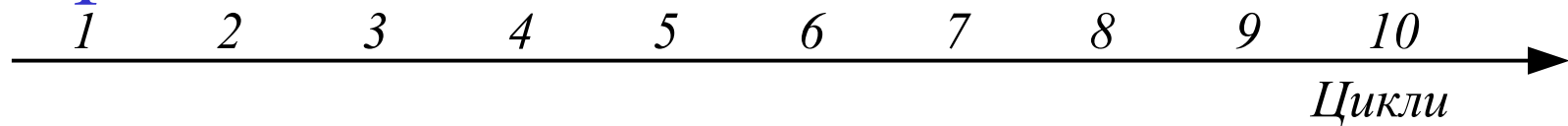
Контролни опасности

Избягването на тези конфликти е трудно. Единственият начин е да не се спазва последователността от изпълнение. С други думи изпълнение на инструкции, които не би трябвало да се изпълняват и следователно водят до нарушаване на контролните зависимости. Това е приложимо само когато може да се запази семантиката на програмата.

Този вид изпълнение се нарича **спекулативно изпълнение**.

Суперскаларни Процесори

Суперскаларно изпълнение



Тип на инструкция

Integer	IF	ID	EX	MEM	WB					
Floating point	IF	ID	EX	MEM	WB					
Integer		IF	ID	EX	MEM	WB				
Floating point		IF	ID	EX	MEM	WB				
Integer			IF	ID	EX	MEM	WB			
Floating point			IF	ID	EX	MEM	WB			
Integer				IF	ID	EX	MEM	WB		
Floating point				IF	ID	EX	MEM	WB		
Integer					IF	ID	EX	MEM	WB	
Floating point					IF	ID	EX	MEM	WB	

Двоен суперскалар



Даннови опасности при ILP

❖ Локализация на Instruction level parallelism

Много инструкции да се изпълняват паралелно

❖ Хардуерът/Софтуерът трябва да запази редът на изпълнение на програмите

Трябва да се запази редът на изпълнение на инструкциите и да изглежда така че те се изпълняват от последователно и следват сорс кода

❖ Важност на данновите зависимости:

- ❖ Отбелязва възможни опасности, а не задължителни проблеми
- ❖ Определя редът, в който резултатите трябва да се калкулират
- ❖ Ограничава отгоре колко паралелизъм може да се използва

Крайна цел: използване на паралелизма като се запази последователното изпълнение само когато то има негативен ефект върху изпълнението на програмата

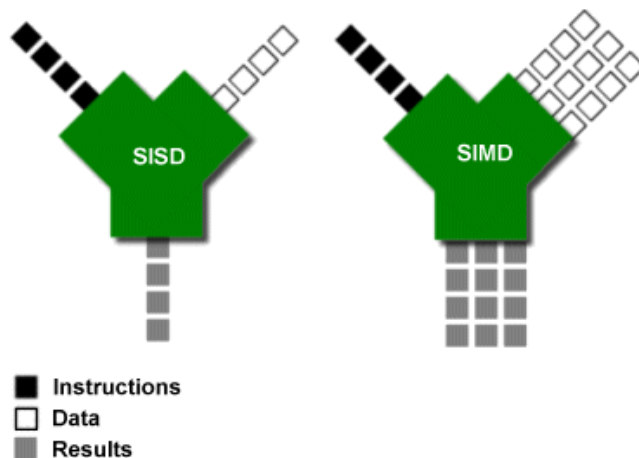
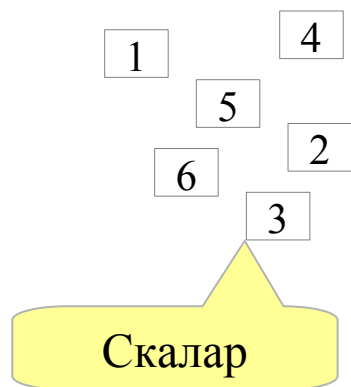
SIMD инструкции

❖ SIMD (Single Instruction Multiple Data)

Техниката се използва, за да се постигне паралелизъм на ниво данни като при векторните процесори.

- ❖ Една от най-новите реализации на SIMD от Интел се нарича AVX-512. Преди нея се появяват: MMX, SSE, ..., AVX, AVX2 и др.

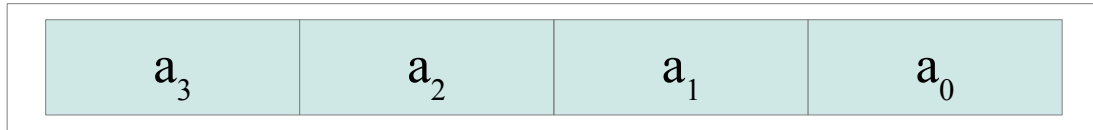
SIMD инструкции



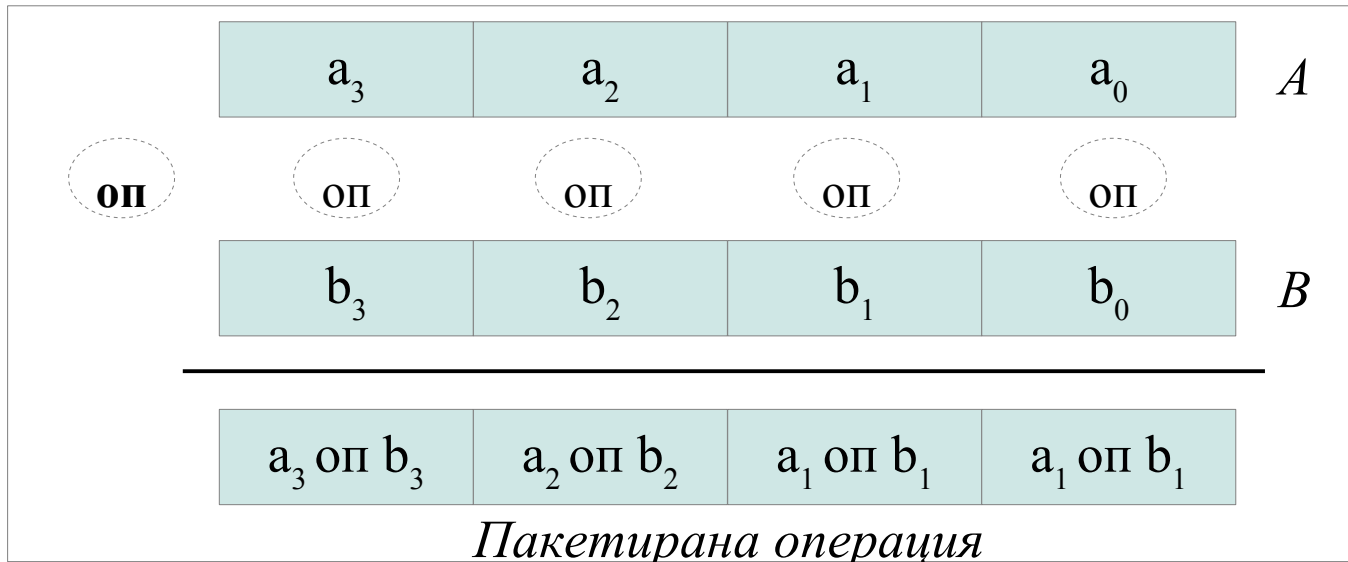
SIMD инструкции

- ❖ Пакетиран тип данни – представлява вектор от няколко скалара

Намират се в отделени специализирани регистри



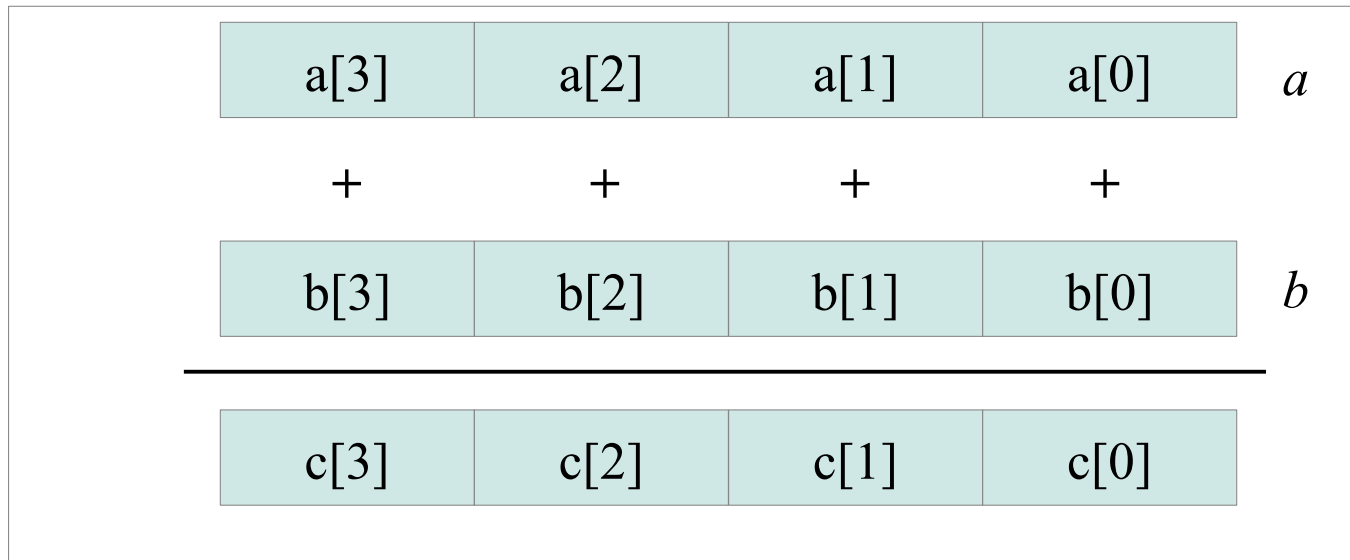
- ❖ SIMD



SIMD инструкции – пример “събиране”

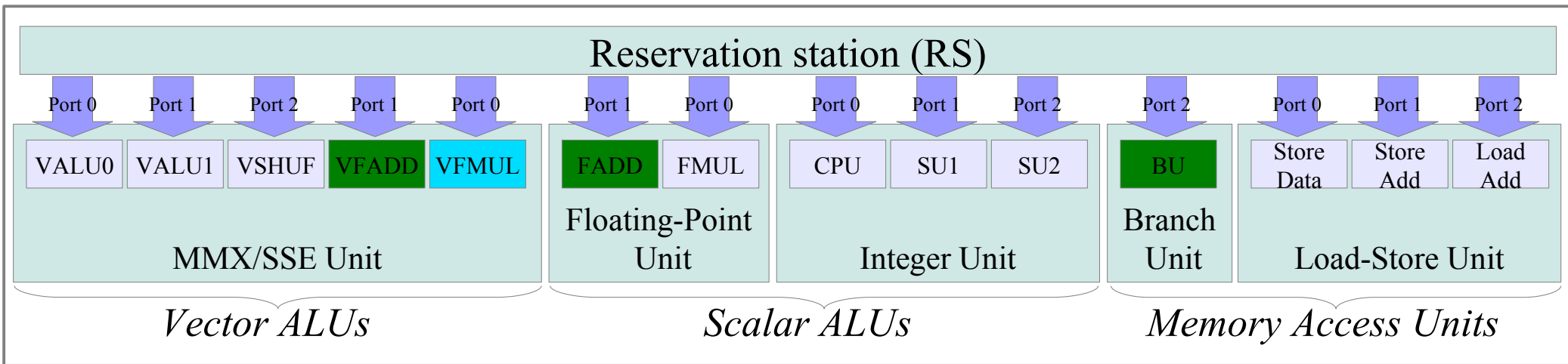
```
for(i=0; i< MAX; ++i)  
  c[i] = a[i] + b[i];
```

❖ SIMD



Intel Core Микроарх. Суперскаларно изпълн.

- ❖ 3 Load/Store
- ❖ 1 Branch
- ❖ 3 Integer
- ❖ 2 Floating point
- ❖ 5 SSE



Спекулативно изпълнение

Различни механизми за предсказване

Постига се по-добра ILP като се преодоляват контролните зависимости чрез спекулиране на резултата от преходите и изпълнява програмата сякаш предсказанията са верни

❖ Предсказване на преходи

Предвиждане на условни, безусловни и изчислими преходи, както и на извикване и връщане от подпрограми. В повечето случаи с много голяма вероятност за успех евъзможно да се предскаже посоката на прехода, както и много често адреса на прехода

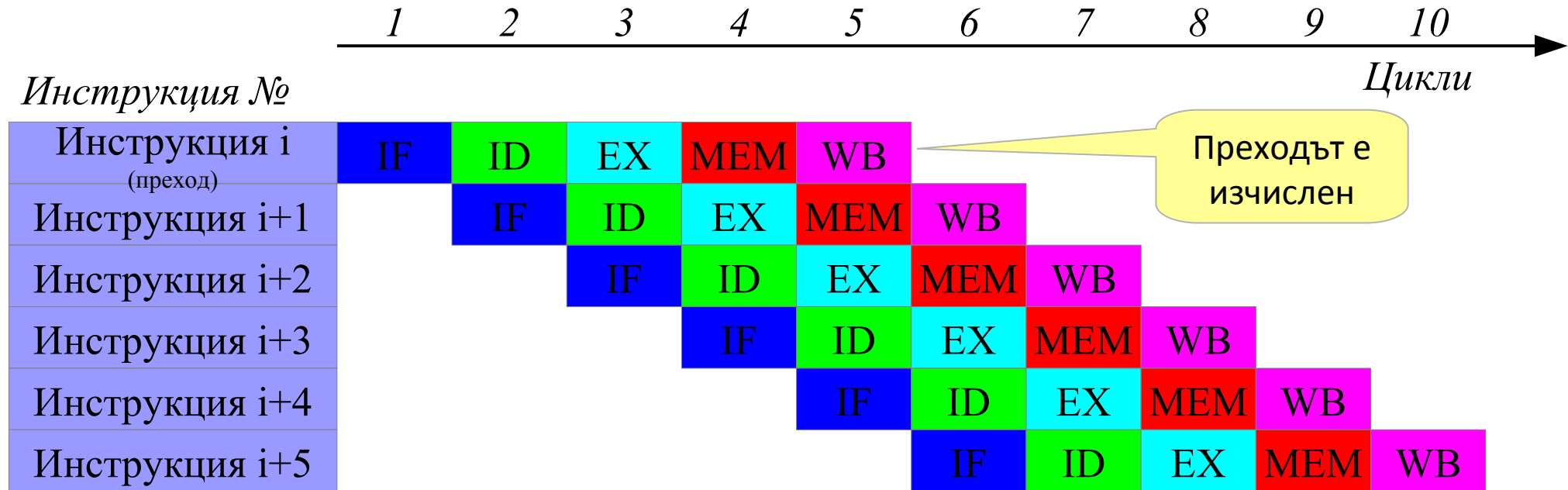
❖ Предсказване на стойности

Предвиждане на стойностите на регистри и други

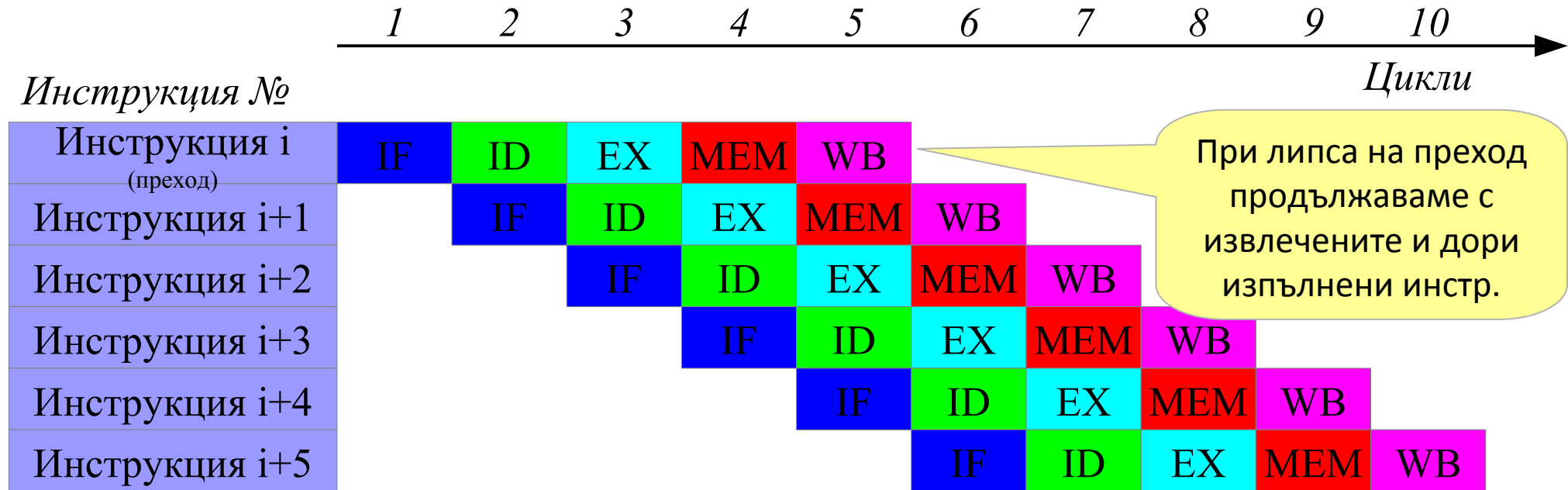
❖ Предварително извличане (Prefetching)

Предвиждане на начинът на достъп до паметта. Повечето достъп до паметта не е напълно хаотичен. Различните инструкции следват накакви очаквани шаблони, например достъпа до стека е предишните/следващите елементи спрямо SP регистъра (Push записва в предишните, Pop извлича от следващите). Това предвиждане дава възможност за предварително извличане на данните от очакваните адреси наведнъж (предварително)

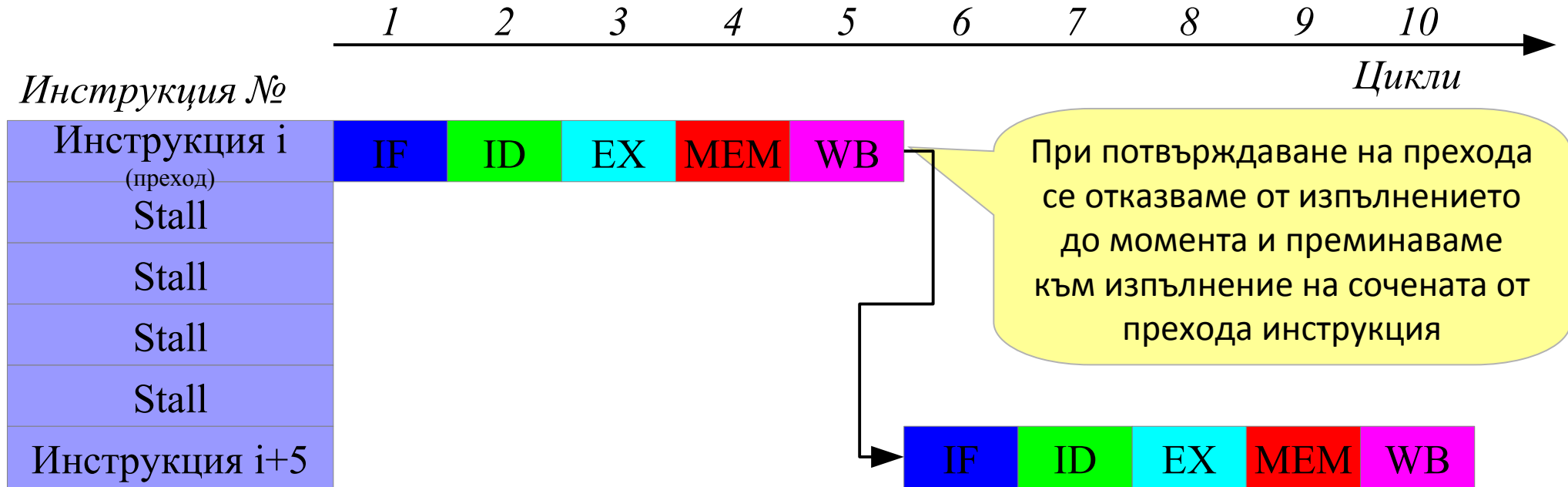
Предсказване на преходи и спекулативно изпълн.



Предсказване на преходи и спекулативно изпълн.



Предсказване на преходи и спекулативно изпълн.



Предсказване на преходи и спекулативно изпълн.

Механизмът за предсказване трябва да определи кое от двете разклонения на условието ще се изпълни

❖ Съвременните предсказващи системи са 99+% точни

Дори могат да предсказват адресите от индиректните/изчислими преходи.

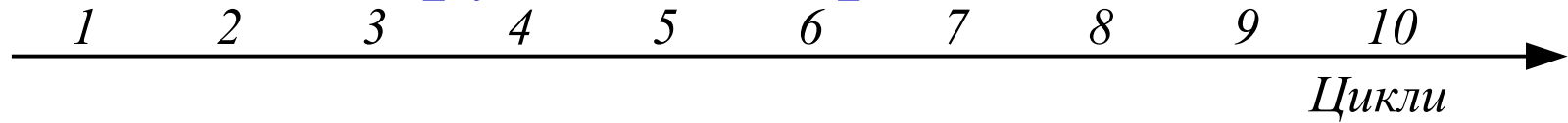
Понеже грешно предсказаните преходи са много малък процент, то печалбата от правилно предсказаните преходи е много по-голяма от загубата при грешно предсказание

Извличат се и се изпълняват спекулативно предсказаните адреси

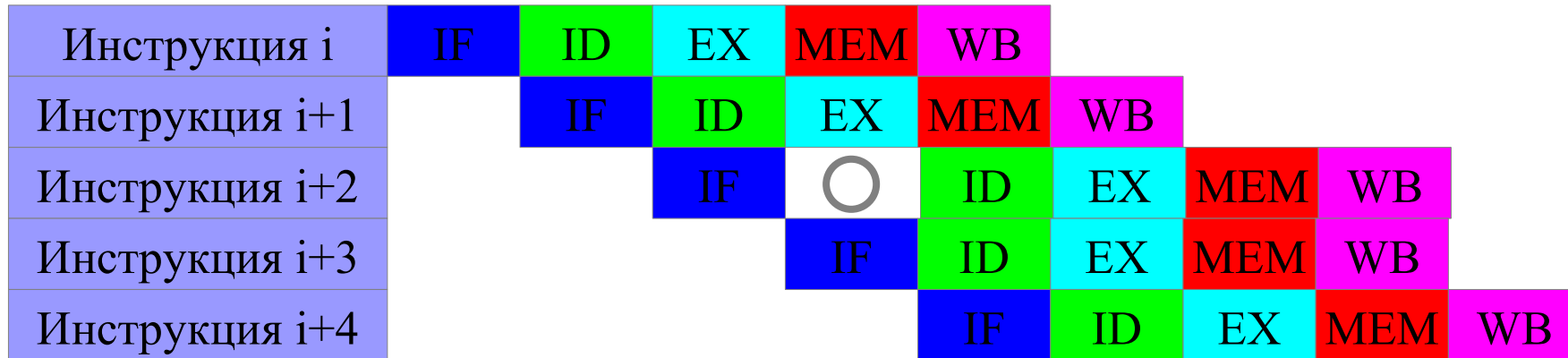
❖ Няма “мехурчета” в конвейера

Когато преходът най-накрая е оценен и адресът е ясен тогава спекулативното изпълнение се потвърждава или отхвърля

Изпълнение на инструкции – при “опасност”



Инструкция №



При невъзможност да се изпълни даден етап от инструкцията, се отлага изпълнението му чрез вмъкване на “мехурче” в конвейера т.е. NOP операция.

Векторни Процесори

Векторни процесори

В суперкомпютъра Cray-1 за първи път успешно се реализира концепцията за векторни процесори.



Векторни процесори

- ❖ Основната “печалба” не е от намаление брой декодирания на инструкции, а идва от скриване на латентността на достъпа до паметта поради извличане на големи обеми данни от последователни адреси в паметта;
- ❖ Изпълняват SIMD векторни инструкции върху вектори с произволна дължина (или по-често с вектори с фиксирана дължина);
- ❖ Някои реализации изпълняват инструкции от вида Памет-Памет, но по-успешните подходи са с използване на големи векторни регистри;

Dataflow Архитектура

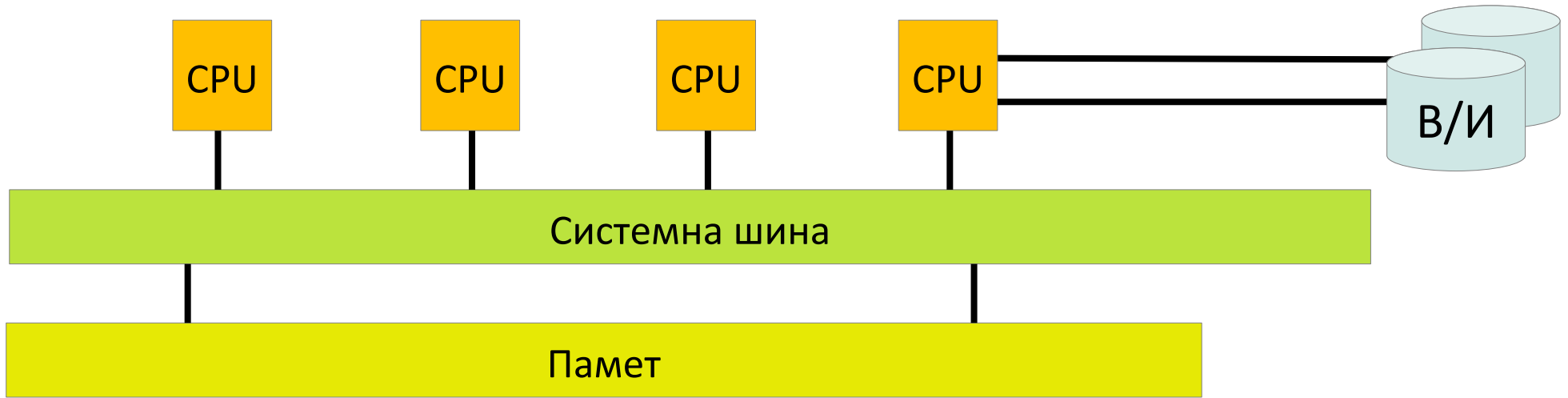
Dataflow архитектура

- ❖ В класическите dataflow архитектури последователността на изпълнението се базира на вида на параметрите (данните), вместо на последователността на контрола. Това изисква ефективна реализация на асоциативна памет и води до недетерминираност на изпълнението;
- ❖ За сега няма успешна реализация на хардуер базиран на тази архитектура, но има много примери на специализиран хардуер, като например Digital Signal Processing, Network routing, Graphics Processing и др.;
- ❖ Поради силно паралелната природа на dataflow подхода той се прилага в много dataflow базирани езици за програмиране;

Мултипроцесорни Системи (Symmetric/Asymmetric)

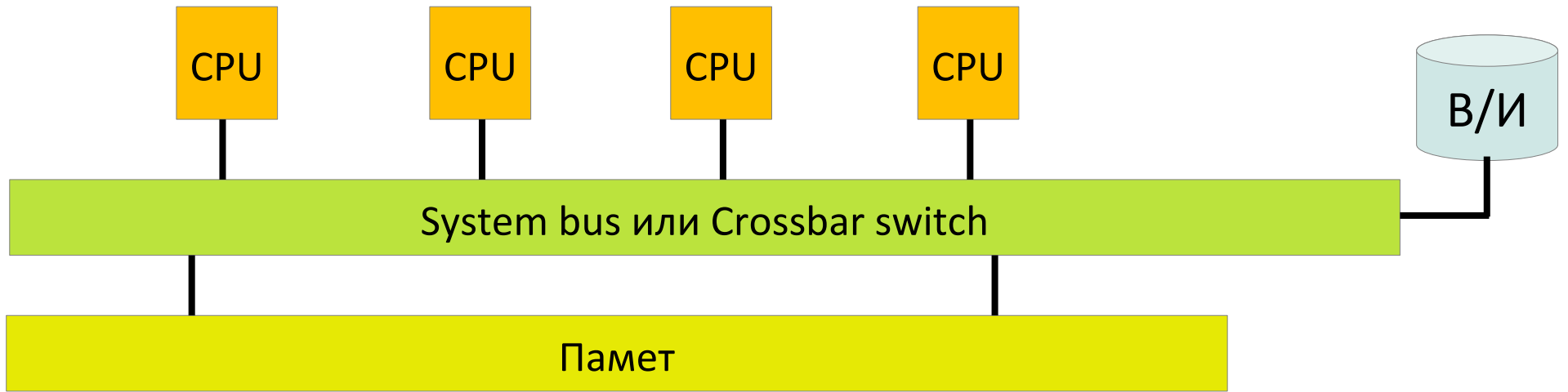
Мултипроцесорни системи

- ❖ Асиметрични – някои от процесорите се различават по предназначението си в системата. Например ОС се изпълнява само на някои процесори и т.н.;



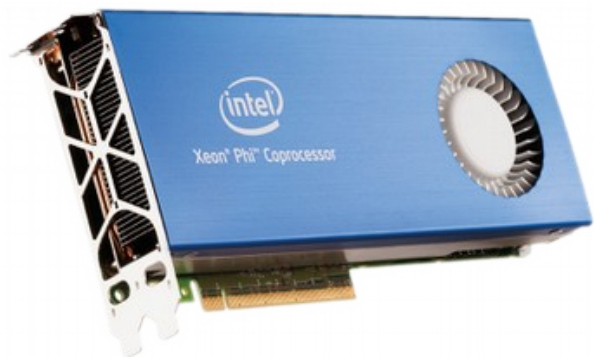
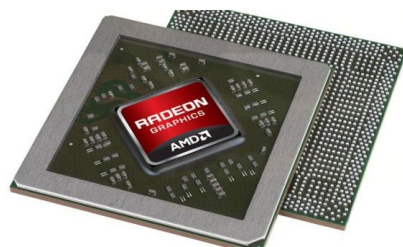
Мултипроцесорни системи

- ❖ Симетрични – всички процесори са “равноправни” в системата;

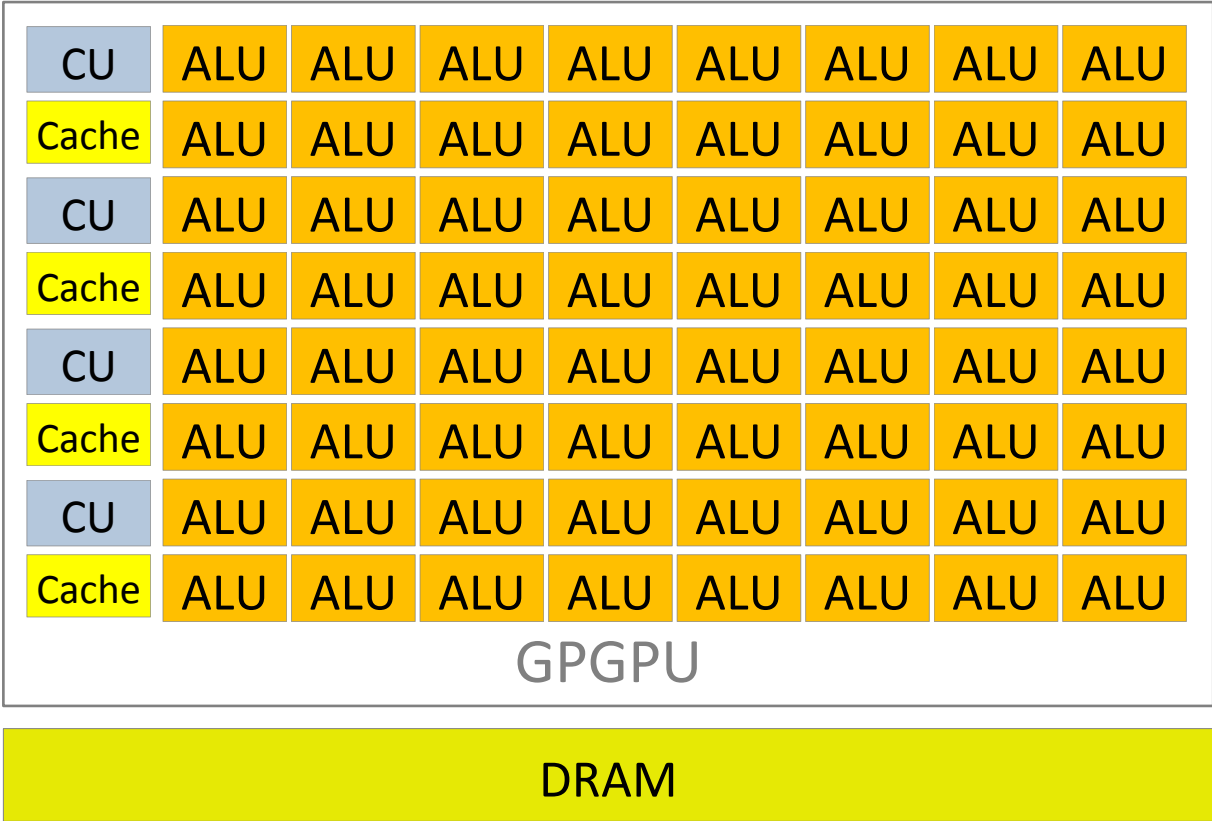
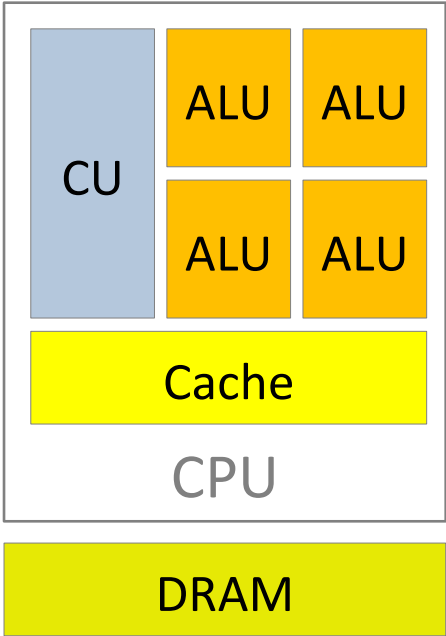


GPU, GPGPU

GPU, GPGPU, ...



CPU vs. GPGPU



CPU vs. GPGPU

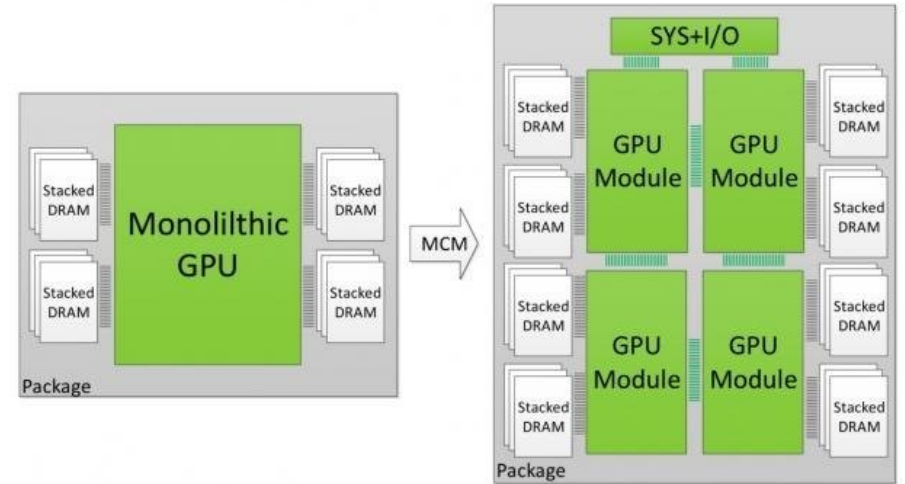
CPU

- ❖ Има нужда от много памет;
- ❖ Скороста е по-ниска;
- ❖ Съдържа малък брой много мощни ядра;
- ❖ Подходящ е за последователно изпълнение;
- ❖ Ниска латентност;

GPGPU

- ❖ Използва по-малко памет;
- ❖ Много висока скорост;
- ❖ Състои се от много на брой “леки” ядра;
- ❖ Създаден е за силно паралелно/стрийм изпълн.;
- ❖ Висока пропускливост;

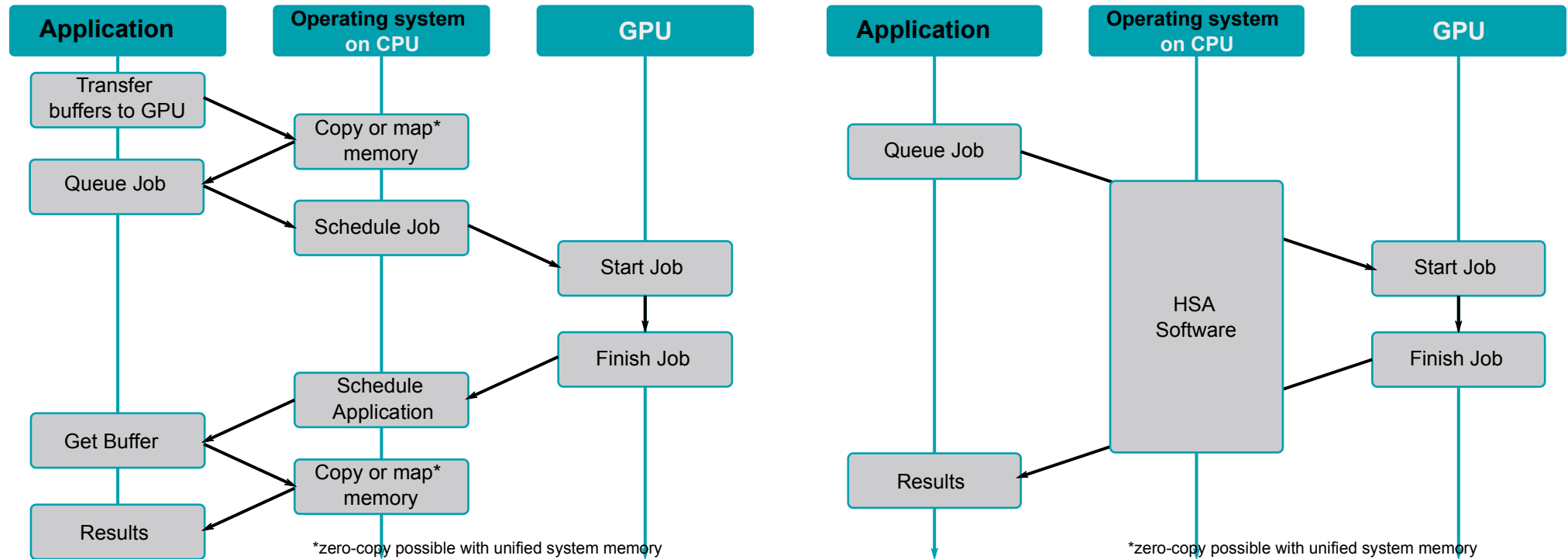
NVIDIA Next Gen-GPU Hopper



Новата архитектура на nVidia Hopper

Хетерогенна Архитектура (HSA)

Heterogeneous System Architecture (HSA)



Високо Производителни Компютри (HPC)

Високо Производителни Компютри (HPC)

IBM Summit
суперкомпютър с
производителност
200 petaFLOPS

Процесори:

POWER9 – 9216 бр.

Tesla V100 – 27648 бр.

Памет:

600GB RAM + 800GB NvRAM

Връзка:

InfiniBand EDR



Високо Производителни Компютри (HPC)

IBM Blue Gene/P
суперкомпютър с
164000 процесорни
ядра

