

ARM Cortex-A. Предсказване на преходите.



Автор: доц. д-р инж. Любомир Богданов

Съдържание

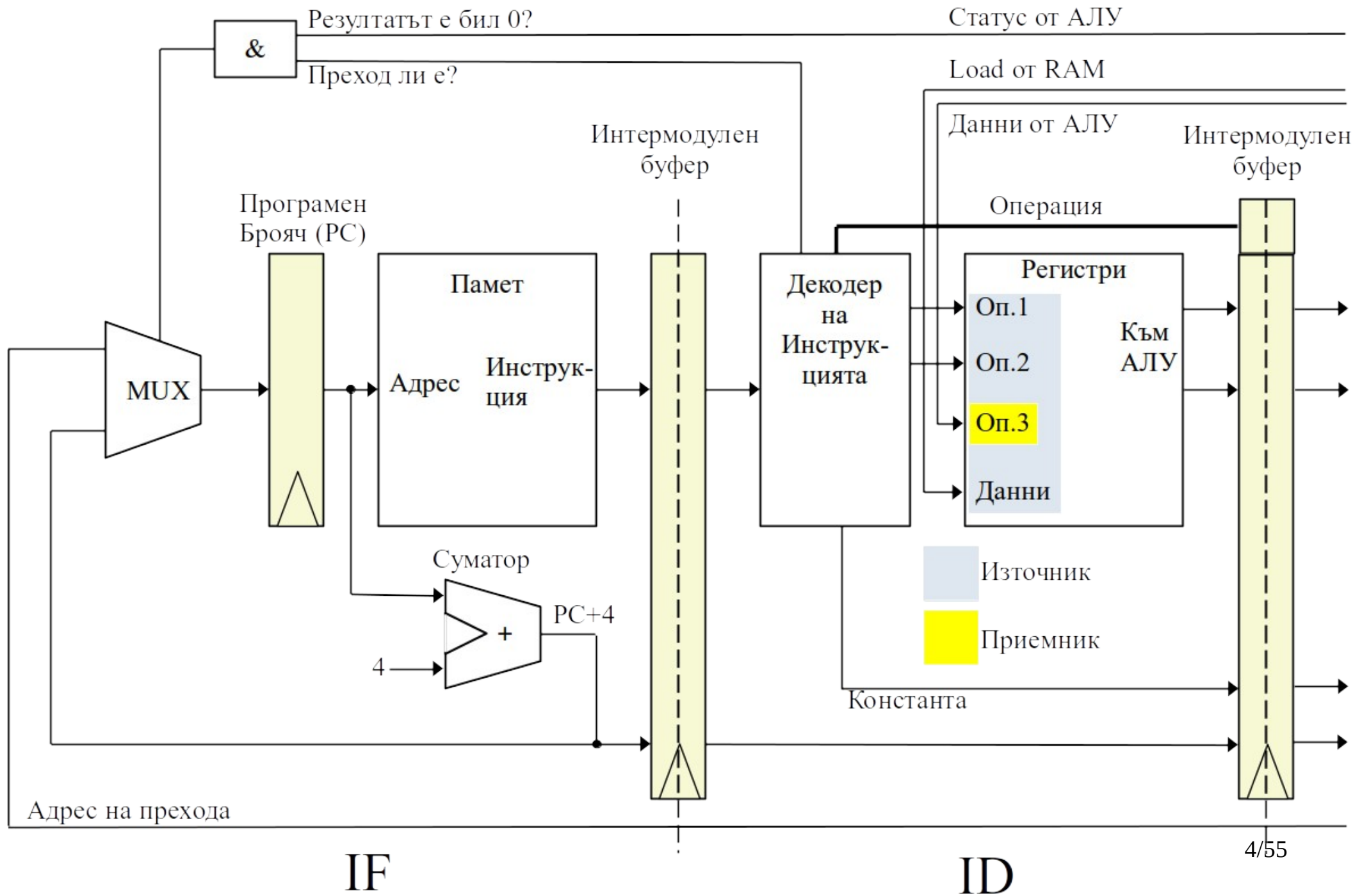
Въведение

1. Контролни опасности (Control/Instruction Hazard)
 2. Статично предсказване
 3. Динамично предсказване
-

ARM Cortex-A реализации:

4. Динамично предсказване на преходите

Контролни опасности



Контролни опасности

Проблем

Инструкциите за преход представляват контролна опасност (**control hazard**) за конвейера на един микропроцесор.

Ако преходът тръгне в друга посока, а не на следващия адрес (след инструкцията за преход) от програмата, конвейера на μ PU трябва да бъде занулен (**flush**).

Зануляването ще доведе до спадане на еквивалентната производителност (заради т.нар. **pipeline bubble**).

Контролни опасности

Възможните решения на проблема са:

- * да се предположи, че **преходът не е взет**;
- * да се детектира прехода възможно **най-рано в конвейера** (възможно в най-първите модули на μ PU);
- * да се отложи прехода (**delayed branch**);
- * да се използва предсказване на преходите (**branch prediction**).

Контролни опасности

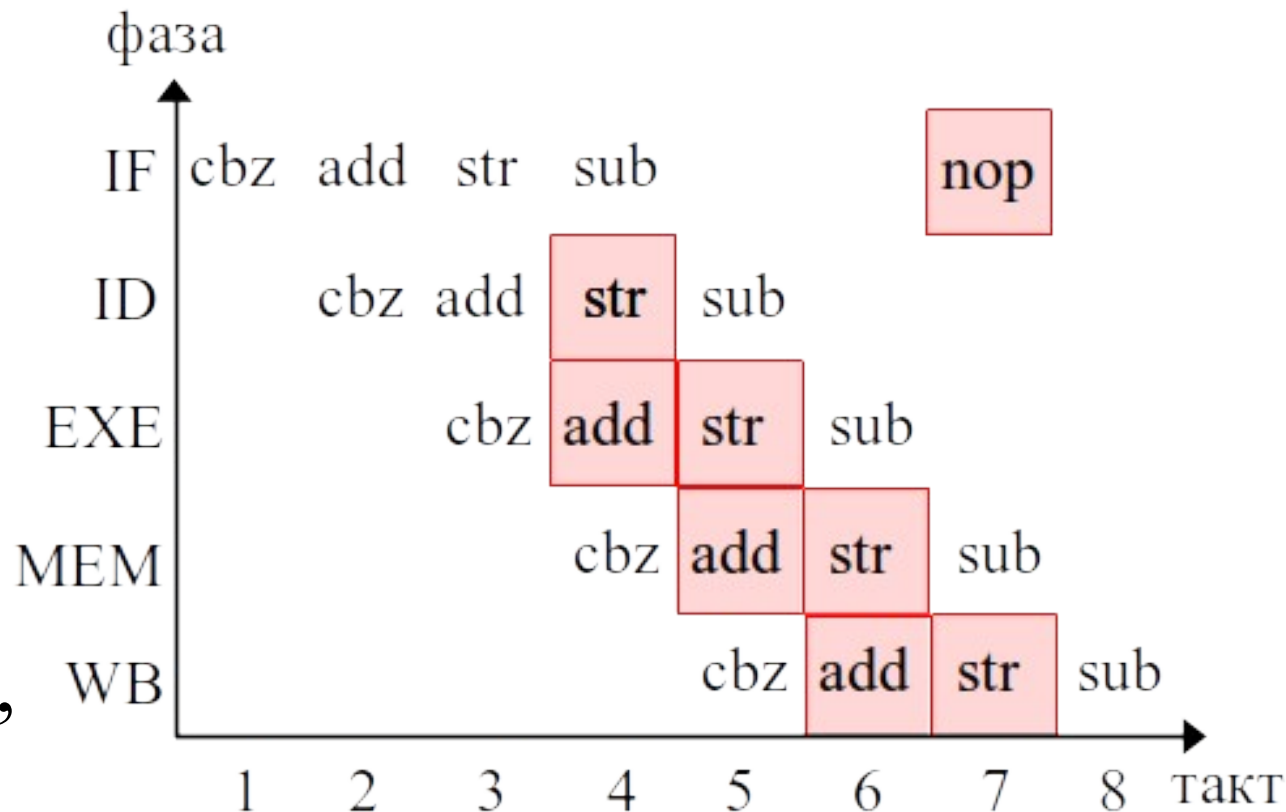
Предполага се, че
преходът не е взет

cbz x3, label
add x2, x2, x3
str x4, [x2], #4
label: sub x0, x0, #1

cbz – переход към label,
ако $x3 = 0$.

Условието на cbz се
разрешава чак в EXE.

Предишните инструкции
се заменят с NOP инструк-
ция



Контролни опасности

Всеки път, когато срещнем условен преход, който бъде взет, **ще загубим 2 такта.**

Пример: ако 20 % от програмата са преходи, и 60 % от тях са взети, еквивалентната производителност ще бъде

$$\text{loss} = 2 \text{ такта} * 0.2 * 0.6 = 0.24$$

$\text{CPI} = 1$ за същата програма, но без преходи =>

$\text{CPI}' = \text{CPI} + \text{loss} = 1.24$ за програмата с преходите

Контролни опасности

Детектиране на прехода възможно най-рано в конвейера:

*преместване на **тестването на условието** на прехода във фазата на декодиране (ID)

*преместване на **изчисляването на адреса** на прехода във фазата на декодиране (ID)

*това ще **намали вредното влияние** на прехода от 2 на 1 празен цикъл

Контролни опасности

Това решение може да бъде постигнато чрез:

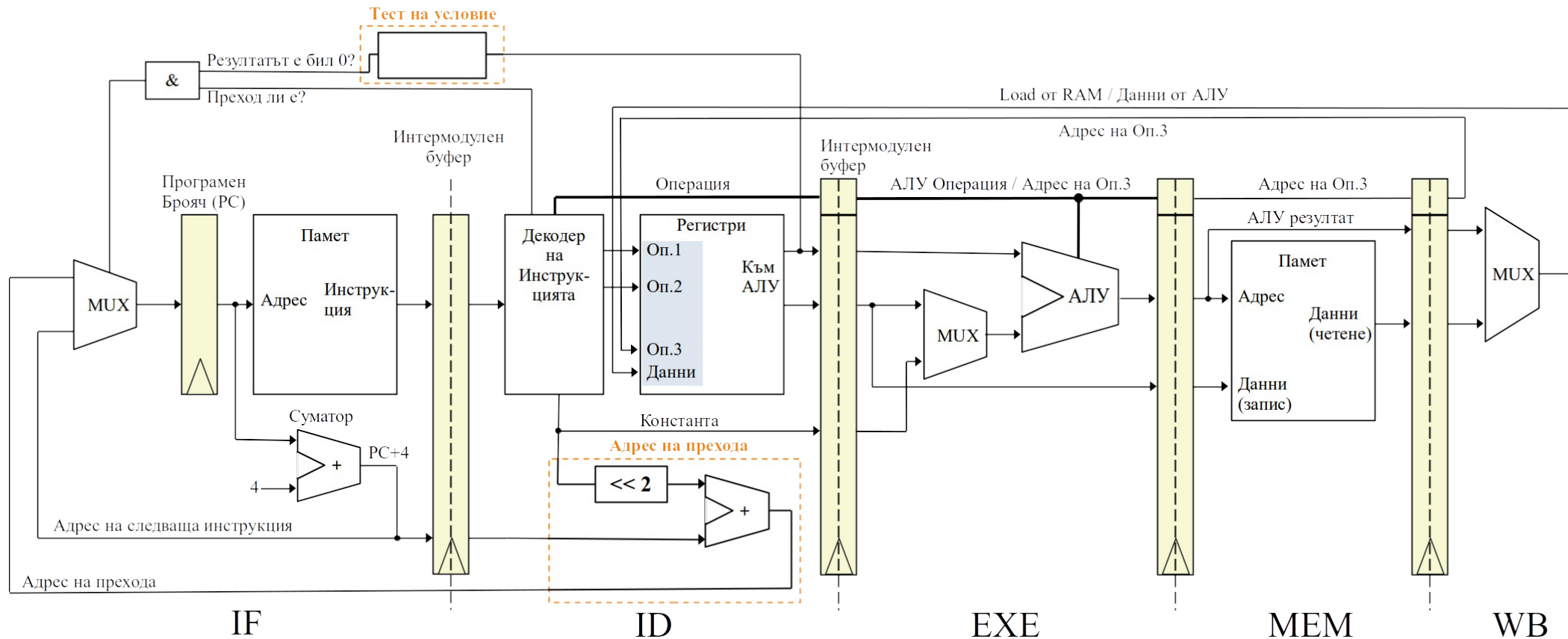
***просто условие за тест:**

- тестването за нулев резултат е лесно
- тестване, изискващо АЛУ е твърде трудно
- тестването не трябва да добавя (много) цикли

***трябва да се следи за даннова опасност (data hazard)**

- ако инструкцията преди прехода записва резултата си в регистъра, който прехода тества, трябва да изчакаме един такт (докато резултатът бъде записан)
- ще са необходими магистрали за забърване (forwarding path) от EXE и MEM степените на конвейера

Контролни опасности



Контролни опасности

Отлагане на прехода (branch delay):

- *предполага се, че преходът е взет и винаги се изпълнява **инструкцията след прехода** (на следващия адрес);
- *в тези случаи, следващата инструкция се нарича **слот за отлагане на прехода (branch delay slot)**;
- *в по-дълги конвейери може да има повече от един слот за отлагане на прехода;
- *load delay slot [b] — аналогични, но след load инструкция.

Пример [b]: MIPS, x86, SPARC
ARM — нямат такъв слот!

Контролни опасности

Процесор **без** branch delay slot

...
...
add r2, r2, 1
sub r3, r3, 1
cbz r3, **label**

...
...
...

label: ←

mov r4, #2
mul r3, r3, r4
...
...

Процесор **с** branch delay slot

Преходът ще отнеме 1 такт по-малко


...
...
add r2, r2, 1
sub r3, r3, 1
cbz r3, **label**

mov r4, #2

...
... Тук r4 не трябва
... да се използва
...

label: ←

mul r3, r3, r4
...
...
Тук r4 може да
се използва

 Branch delay slot

Контролни опасности

***В слота за отлагане на прехода компилаторът може да сложи инструкция, която няма общо с кода на прехода.**

***На практика компилаторът успява да запълни слота в 60 – 70 % от случаите [1].**

***Колкото по-дълъг е конвейера, толкова повече инструкции трябва да се сложат в слота и **ТОЛКОВА ПО-ТРУДНО** ще е на компилатора да го запълни.**

***Ако не може да се намери инструкция, която да извърши полезна работа в този слот, трябва да се използва **NOP**.**

Контролни опасности

Проблем: ARM Cortex-A15 има 15 степени на конвейера. Захващане на инструкции от **грешния клон** при преход **би отнело 14 такта**.

Също - на един такт се захващат 4 инструкции и се декодират 3 \Rightarrow зануляването на конвейера при преход ще изтрие >40 инструкции от ядрото, които са вече навлезли в него [1].

Предсказване на преходите (**branch prediction**):

Подходящо за използване в процесори с повече ≥ 3 степени на конвейера.

***Статично** (остарял метод)

***Динамично**

Контролни опасности

Статично предсказване†:

*Взема се решение за изхода на прехода в зависимост от това дали целевия адрес (branch target address, ВТА) е **напред** или **назад** в програмата.

*Преходи **назад** често са част от цикли \Rightarrow в повечето случай биват взети \Rightarrow предсказваме:

“Ако $ВТА < РС$, значи прехода ще бъде взет”

*Типична успеваемост: 65 % [1]

Пример: ARM10 (ARMv5TE, от 2000 г., 250k транз.)

†Static branch prediction

Контролни опасности

Динамично предсказване на 1 ниво с 1 бит†:

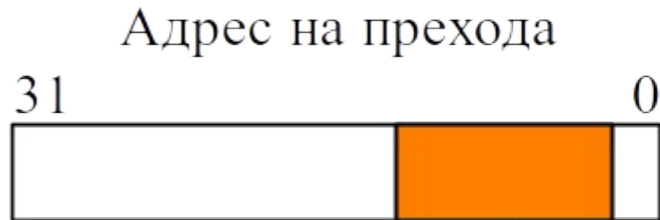
*По време на изпълнението на програмата един буфер с предсказания (branch prediction buffer, BPB) съхранява по **1 бит за всеки преход** от програмата. Този бит съдържа информацията: **преход взет / преход невзет** [с].

*Ако предсказанието се окаже грешно, конвейера се занулява и съответния бит от BPB се преобръща.

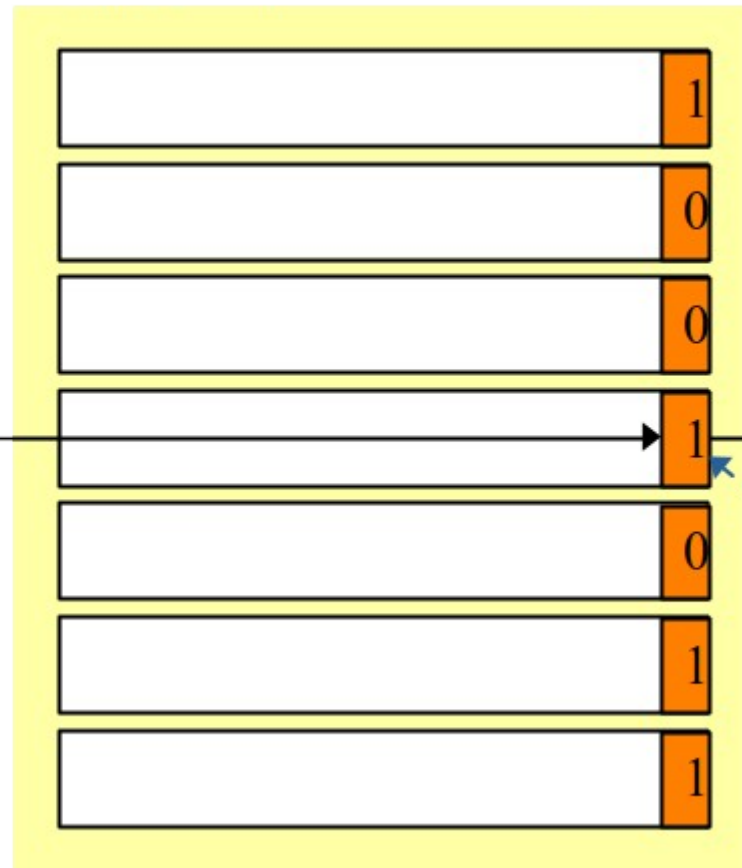
†One-level branch prediction with 1-bit buffers

Контролни опасности

Буфер с предсказания (BPB)



На един адрес на преход от програмата се съпоставя един бит от BPB. Всеки бит съдържа изхода на прехода (т.е. взет / невзет) от предишното изпълнение на този преход. Индексът на бита от BPB блока е свързан с част от адреса на прехода (оранжево).



Предсказание

Обновяване от EXE

Проблем: битът се обновява при всяко невярно предсказание. Цикъл от 10 итерации с 10 прехода ще има 2 грешни предсказвания (на първата и на последната итерация).

Контролни опасности

Динамично предсказване на 1 ниво с 2 бита†:

Въвежда се хистерезис – използва се двубитов брояч.

Необходим е по един реверсивен брояч с насищане за всеки елемент от ВРВ.

Числото на брояча се обновява на всяко грешно предсказване.

Старшият бит на брояча е битът на предсказването => сменя се на всеки две грешни предсказания.

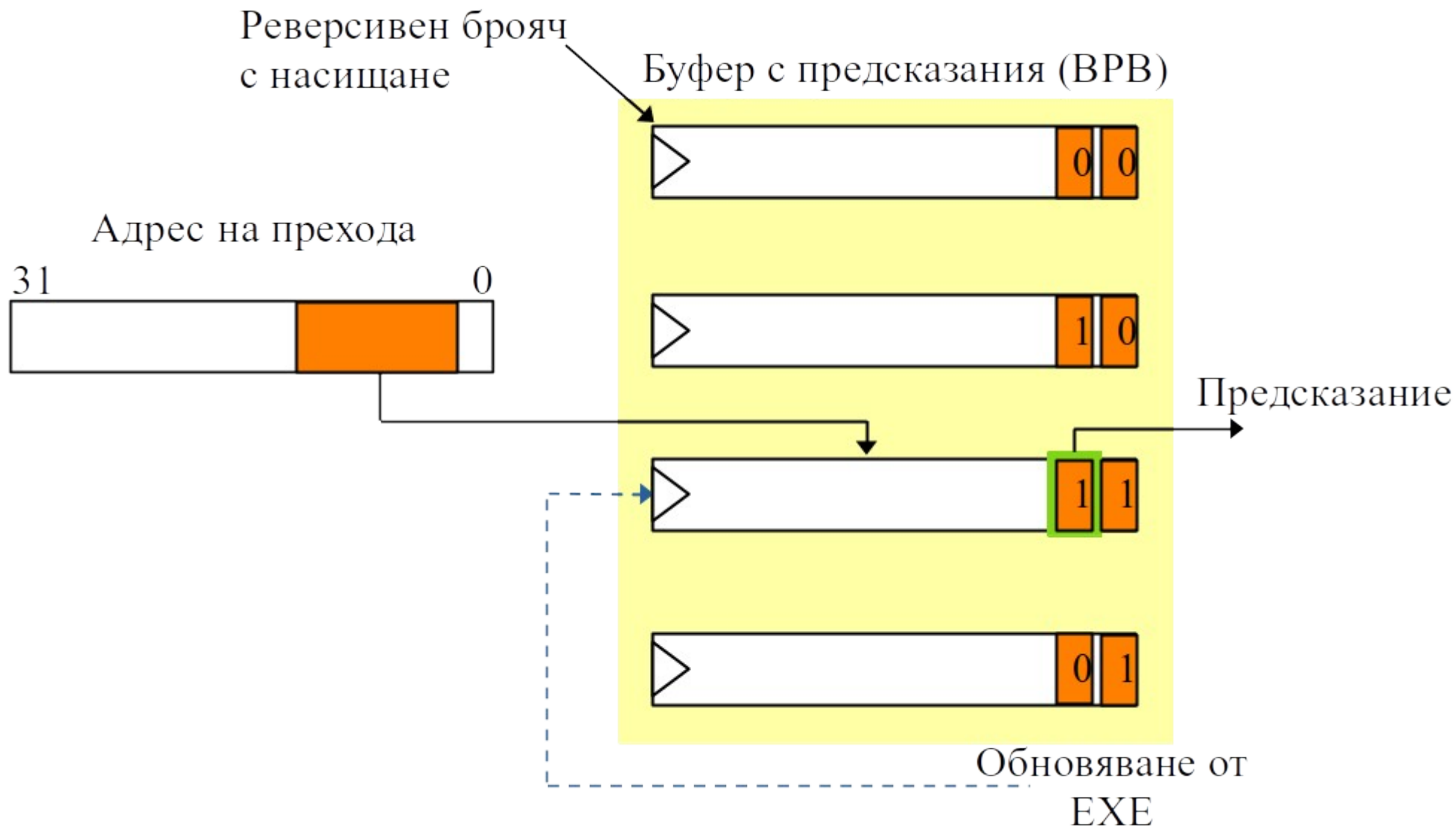
Контролни опасности

```
if(taken){  
    count++;  
    if(count == 4){  
        count = 3;    //насищане  
    }  
}  
else{  
    count--;  
    if(count == -1){  
        count = 0;    //насищане  
    }  
}
```

Контролни опасности

Брояч	Вероятност (англ. ез.)	Вероятност (бълг. ез.)
00	Strong not-taken	Силно невзет
01	Weak not-taken	Слабо невзет
10	Weak taken	Слабо взет
11	Strong taken	Силно взет

Контролни опасности



Контролни опасности

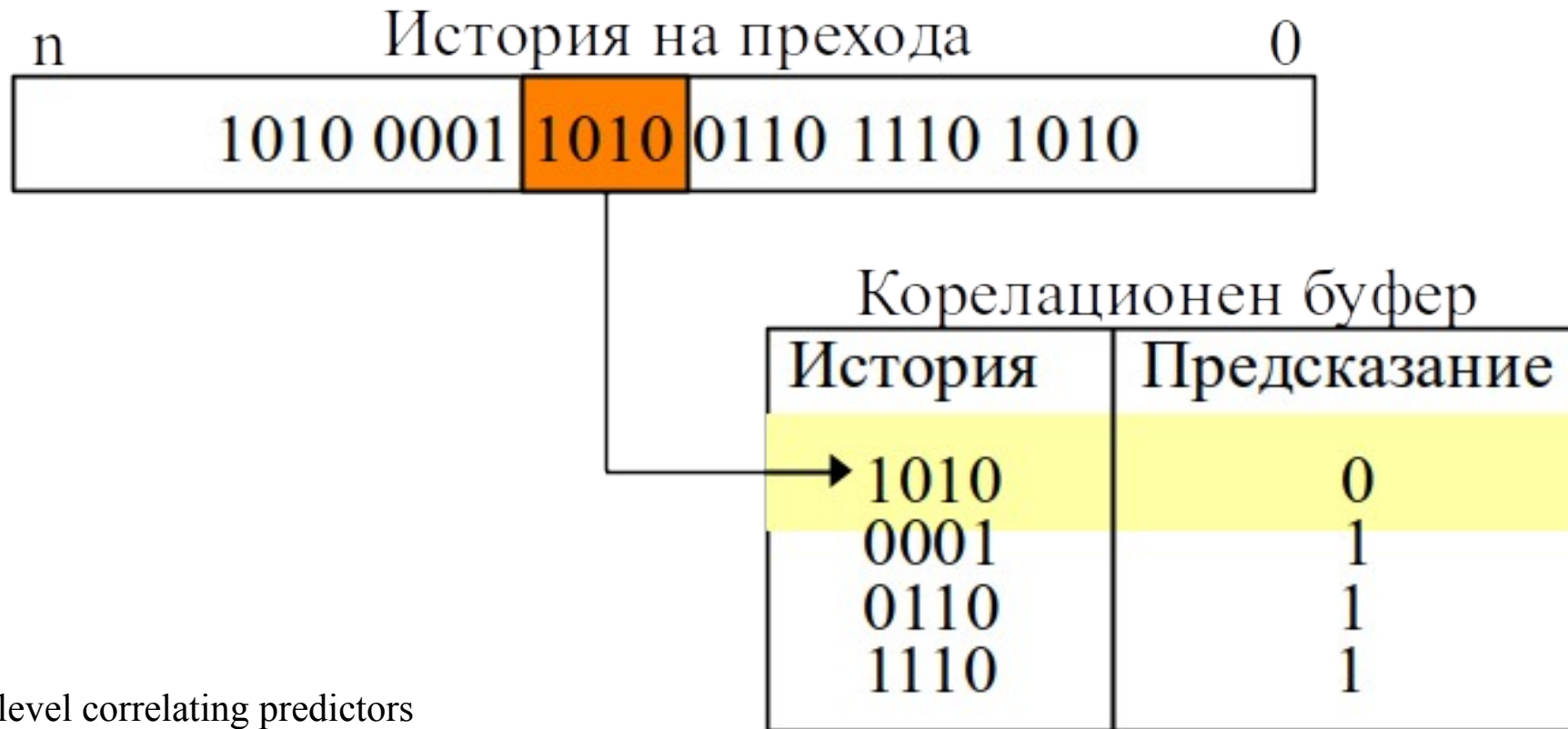
- * Често изхода на **един и същи преход** е свързано с миналото му (локална история).
- * Често изхода на **един преход** е свързано с миналото на **съседни преходи** (глобална история).
- * Може да се направи **схема на индексирание**, която е по-сложна от просто взимане на младши битове от адреса на прехода и включва локалната и глобалната история на прехода. Това ще повиши процента на **правилни предсказания**.

Контролни опасности

Динамично предсказване с корелационен буфер†:

*Взема се под внимание историята на прехода.

*Това означава, че трябва да се съхранят битовете от **изхода на един преход** в един дълъг буфер и да се анализират с “движещ се прозорец”.



†Two-level correlating predictors

Контролни опасности

Динамично предсказване с корелационен буфер†:

*Ако схемата се доразвие, може да се вземе под внимание **историята на преходите преди настоящия преход** => много често те са свързани:

```
if(a == 0){          //Преход 1
    b = 1;
}
if(b == 1){          //Преход 2
    c = (d + e)/2;
}
else{
    c = (d-e)/2;
}
```

†Two-level correlating predictors

Контролни опасности

Реализацията предсказващ модул с корелация включва следните елементи:

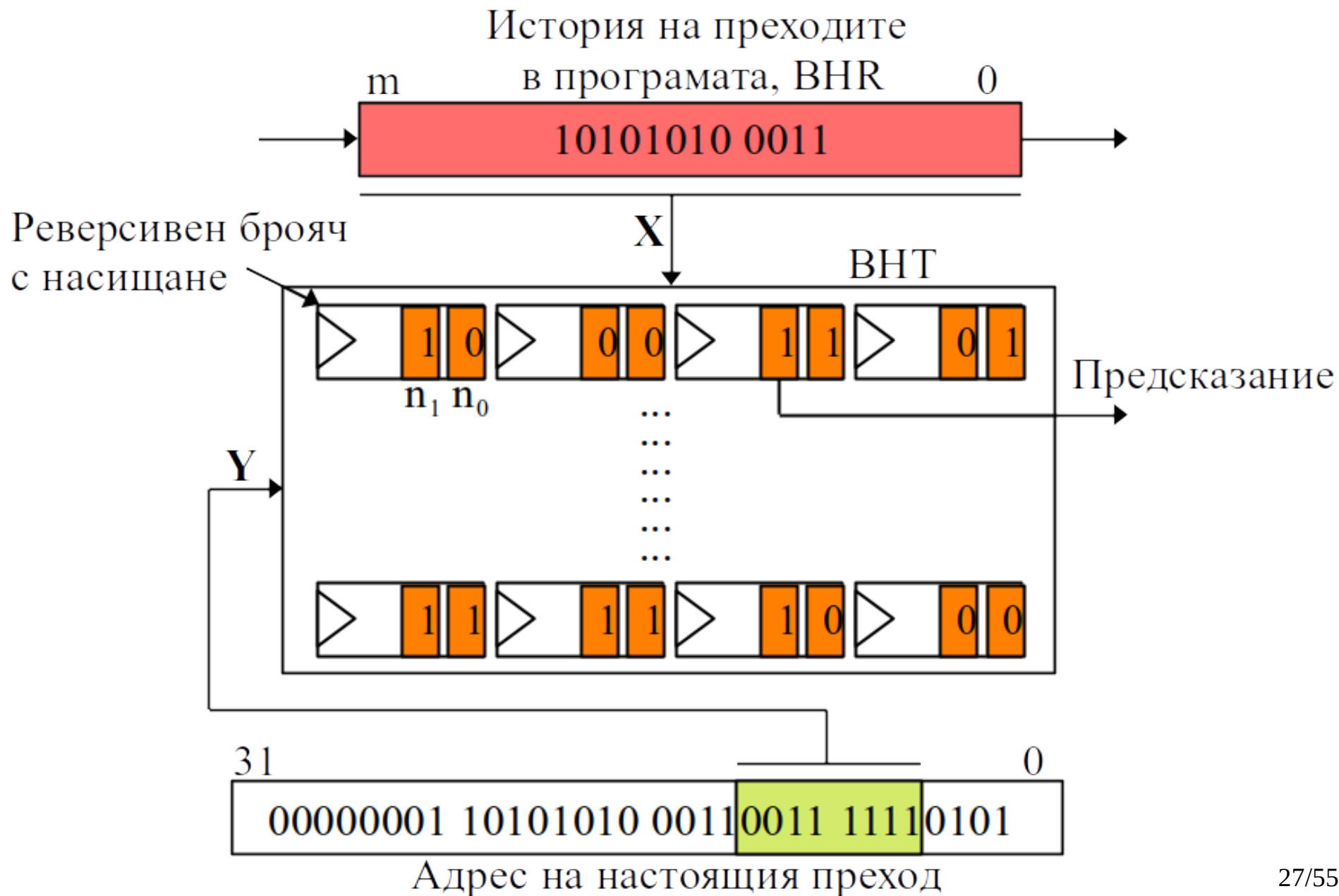
- *(преместващ) регистър с история на преходите (branch history shift register, или само branch history register, **BHR**) [7]

- *таблица с n-битови броячи с насищане (branch history table, **BHT**)

- *Предсказвател† от вида (m, n) означава, че се съхранява информация за m прехода, като за всеки преход се използват n-битови броячи с насищане [6].

†Branch predictor

Контролни опасности



Контролни опасности

Не само е важно да се знае дали дадена инструкция е **преход** или не, както и дали е **взет** или не, но също и **къде в паметта** се намира прехода.

Затова:

Буфер за целеви адреси, ВТВ[†], ВТАС[‡] – буферна памет за съхранение на целевите адреси на преходите. Някои такива памети съхраняват и инструкцията (target instruction), която ще се извлече от този адрес. Това означава, че ако се окаже, че предсказанието е било правилно, тактовете на прехода ще са = 0 [1] (инструкцията за переход се заменя с target instruction).

28/35

Контролни опасности

Кеш памет за целеви инструкции, BTIC† – буферна памет за съхранение на целеви инструкции (повече от една), разположени на предсказания целеви адрес на прехода.

†Branch target instruction cache

Контролни опасности

Стек за адреси на връщане† – буферна LIFO памет за съхранение на целеви адреси на връщане, която подобрява предсказанието на преходите.

ВНИМАНИЕ! Да не се бърка с hardware stack, напр. на Microchip PIC18, Intel 8087 FPU).

Понеже една функция може да бъде извикана от различни части в кода, **инструкцията за връщане** в основната програма **ще сочи към различни целеви адреси**. Това би довело до намаляване на точността на предсказване, използваща ВТВ паметта. Затова този проблем се преодолява с return-address stack.

†Return-address stack

Контролни опасности

ВНИМАНИЕ! Return-address stack също не трябва да се бърка със софтуерен стек.

Софтуерен стек – регион от RAM (т.е. не е част от ядрото), който съхранява състоянието на ядрото, *адресите на връщане от подпрограми* и локалните променливи на подпрограмите. Размерът му е ограничен единствено от обема на RAM.

Динамично предсказване на преходите

Модул за предварително извличане на инструкции (**Prefetch Unit, PFU**) – част от микропроцесорното ядро, изпълнява следните операции [2] [а]:

- ***предполага** (спекулира) кои инструкции ще се изпълнят следващи и ги извлича предварително от паметта;

- ***прави разлика** между **Thumb-2** инструкциите дали са 16- и 32-битови, и ги предоставя на ядрото; помага за изпълнението на IT инструкции;

Динамично предсказване на преходите

***спомага операциите за работа с кеш паметта за инструкции; зареждане на кеш линия; load/store операции с ITSM† - PFU управлява адресната магистрала на i-кеш и ITSM;**

†Instruction Tightly Coupled Memory (ITSM) – съдържа код на прекъсвания/изключение, който трябва да бъде изпълнен бързо и който не би могъл да толерира кеш пропуск, cache miss

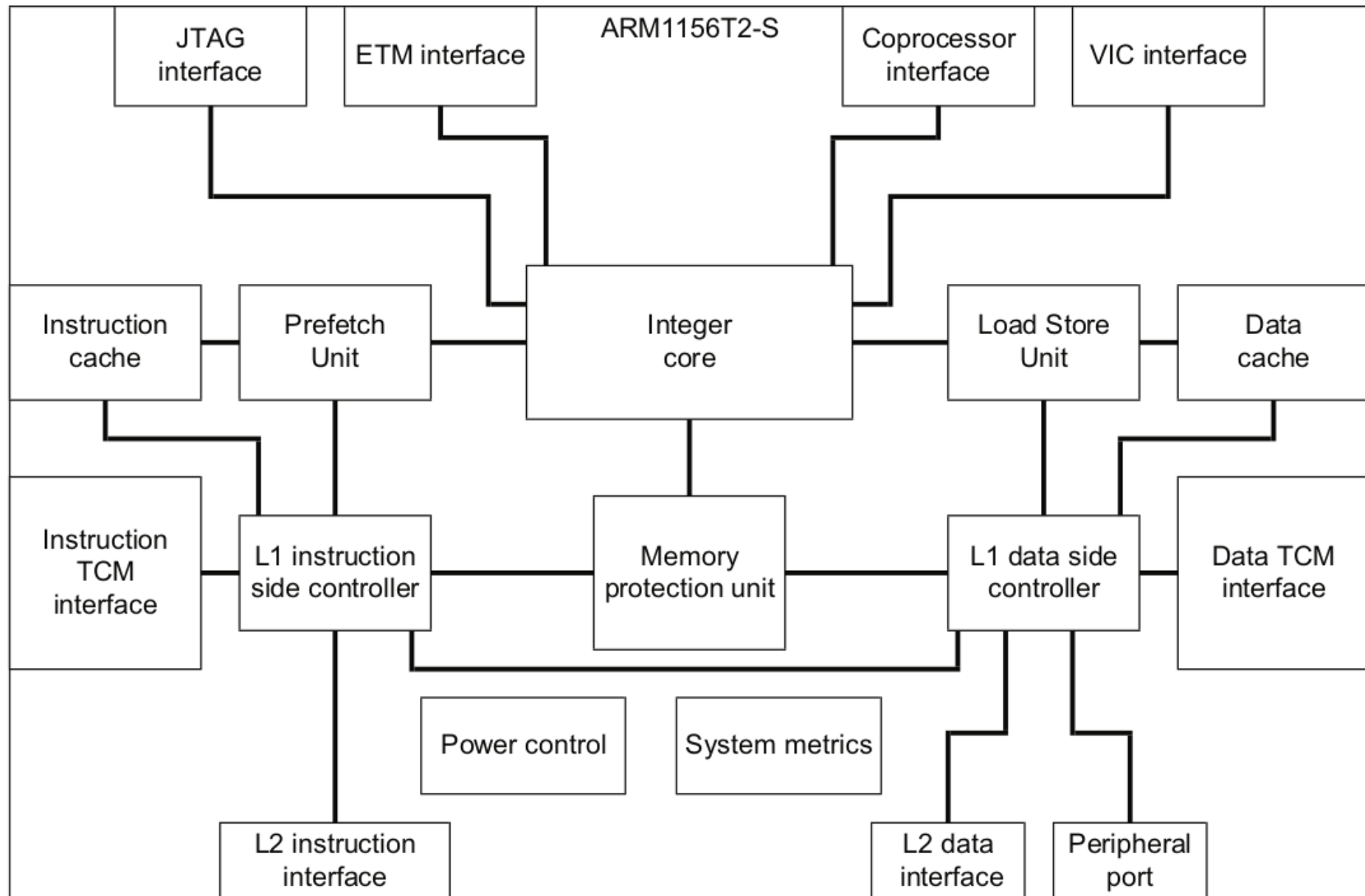
Динамично предсказване на преходите

В режим на извличане на инструкции, паметта може да бъде четена на 2 думи / 4 полу-думи за един такт (многопортова памет).

RFU за класическия ARM1156T2-S може да буферира във FIFO до 5 думи.

RFU е мястото, където се извършва предсказването на изпълнението на програмата [2].

Динамично предсказване на преходите

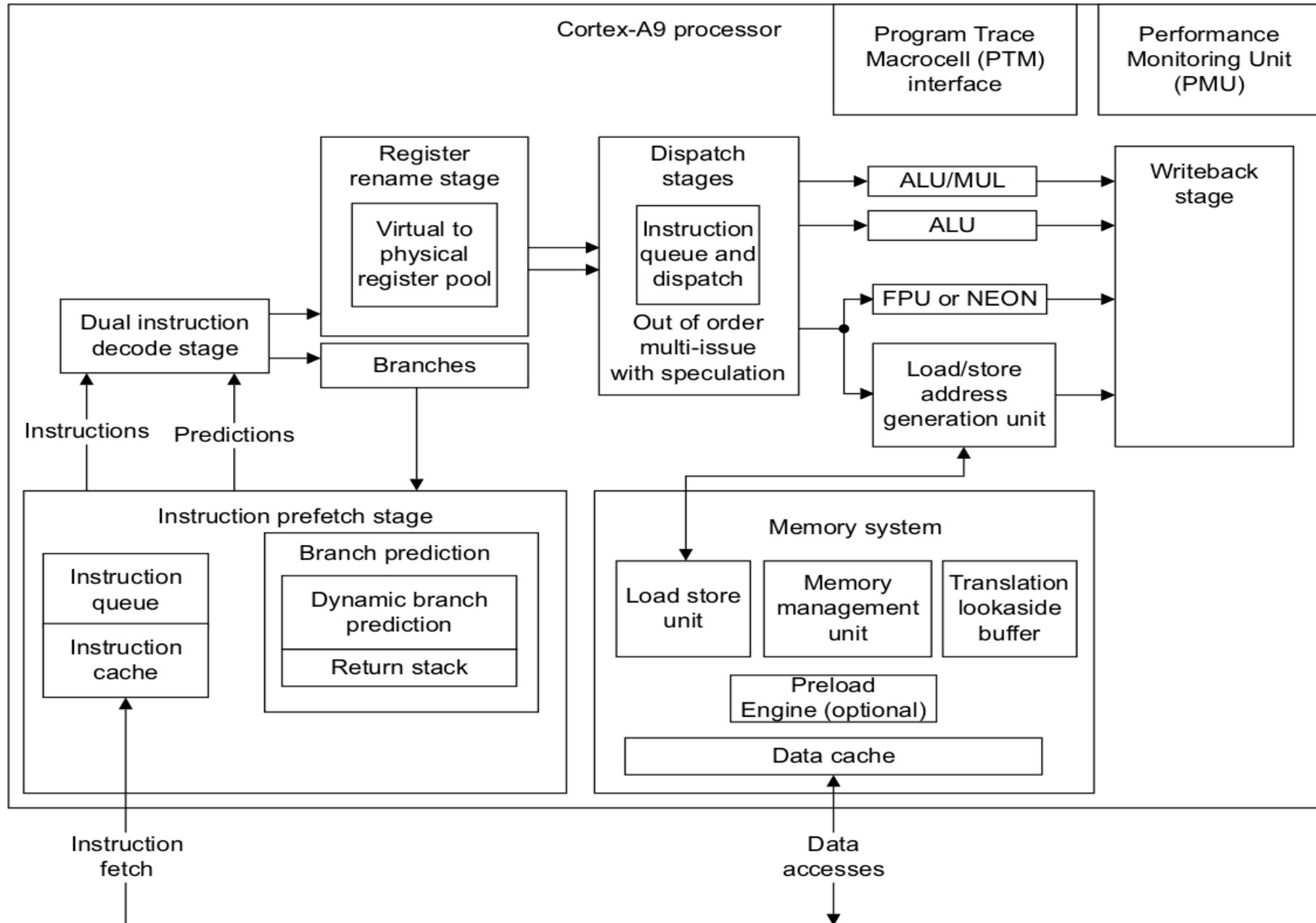


[2]

Figure 1-2 ARM1156T2-S processor block diagram

Динамично предсказване на преходите

Figure 2-1 shows a top-level diagram of the Cortex-A9 processor.



[3]

Динамично предсказване на преходите

Извличането на инструкции от PFU модула се управлява от:

- *програмният брояч PC;
- *модула за предсказване на преходи BP (Branch Predictor);
- *връщане от подпрограма, указано от Return Stack;
- *изключения/прекъсвания, генерирани от основното ядро;

Динамично предсказване на преходите

Предсказването на изпълнението на програмата се състои в:

***предсказване изхода на условните преходи и ако се предвиди прехода да бъде взет, изчислява се адреса, от който да продължи извличането на програмата;**

***предсказване адреса на връщане от подпрограма, чрез използването на стек за връщане (Return Stack).**

Основното ядро на процесора изчислява дали предсказанията са били верни [2].

Динамично предсказване на преходите

Предсказването на преходите е необходимо, защото Cortex-A имат дълги конвейери [4], [5], [b]:

- *Cortex-A15 – 15 степени
- *Cortex-A57 – 19 степени
- *Cortex-A72 – 16 степени
- *Cortex-A9 – 9 до 12 степени
- *Cortex-A5 – 8 степени
- *Cortex-A7 – 8 степени
- *Cortex-A8 – 13 степени
- *Cortex-A12 – 11 степени

и зануляването (flush) им ще намали еквивалентната производителност (CPI) на съответните ядра.

Динамично предсказване на преходите

В ARM процесори, които нямат PFU, се разбира накъде ще тръгне програмата чак след като инструкцията за **разклонение** мине през EX модула.

В такива процесори се **предполага**, че **преходът не е взет** и конвейера се пълни с инструкции, които се намират след инструкцията за преход.

Това означава, че:

- *преход, който не е взет ще отнеме 1 такт

- *преход, който е взет, ще отнеме >1 такт (колкото са степените), поради зануляването на конвейера

Динамично предсказване на преходите

При връщане от подпрограма, ако преходът бъде разпознат от PFU, процесорът може да е буферирал адреса на връщане в **стек за адреси на връщане** (Return Stack, RS) и PFU да започне да извлича инструкции от правилния адрес, без да се изчаква преходът да мине през EX фазата.

Динамично предсказване на преходите

Размерът на return-address стека е (типично) няколко/няколко десетки думи.

Например [5]:

	Cortex-A5	Cortex-A7	Cortex-A8	Cortex-A9	Cortex-A12	Cortex-A15
Release date	Dec 2009	Oct 2011	July 2006	March 2008	June 2013	April 2011
Return stack entries	4	8	8	8	8	48

Динамично предсказване на преходите

Както е видно от миналия слайд, *адреси на връщане от подпрограми* се съхраняват както в RAM, така и в ядрото – в стека с адреси за връщане (return-address stack).

Ако PFU намери адрес на връщане в този стек, ще използва него за предсказване на прехода (**бързо**). Ако не го намери – адресът ще бъде зареден от RAM (**бавно**).

Динамично предсказване на преходите

Предсказването на преходите може да бъде разрешавано и забранявано.

Това става от копроцесора за системен контрол (CP15), регистър c0.

Алтернативното му име SCTRL [5] – System ConTRoL register.

Управлява:

- *подравняването на паметта
- *endianess-a
- *поведението при прекъсвания
- *адресите на изключенията
- *поведението при Fault събитие
- *кеш памети
- *MMU
- *MPU
- *предсказване

Динамично предсказване на преходите

Figure 4-8 shows the SCTLR bit assignments.

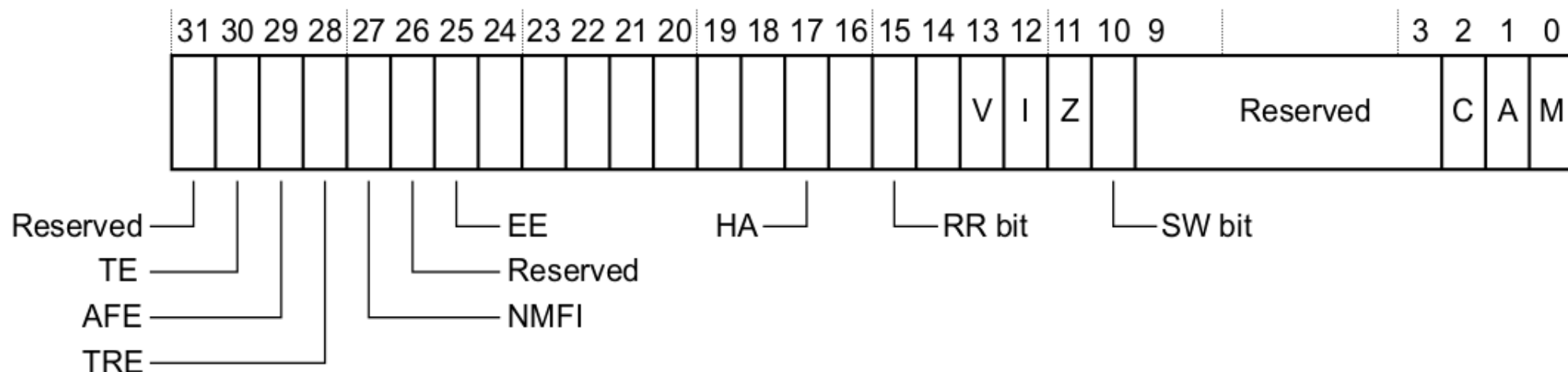


Figure 4-8 SCTLR bit assignments

Z – при 0 предсказването на преходите забранено. Това е състоянието на бита след ресет. При 1 предсказването е разрешено.

Динамично предсказване на преходите

ВНИМАНИЕ! Преди да се разреши пресказването, трябва да се занули **ВТАС** (Branch Target Address Cache) кеша – това е специализирана памет от 512 думи, съдържащи целеви адреси на прехода.

Зануляването на ВТАС става с **инструкция ISB** (Instruction Synchronization Barrier)

Динамично предсказване на преходите

Auxiliary Control Register (ACTRL) от копроцесор #15 - използва се за настройка на предсказването при работа с кеш, контрол на проверка по четност на паметта, разрешаване на L2 кеш, спекулативни достъпи по AXI и др [3].

Динамично предсказване на преходите

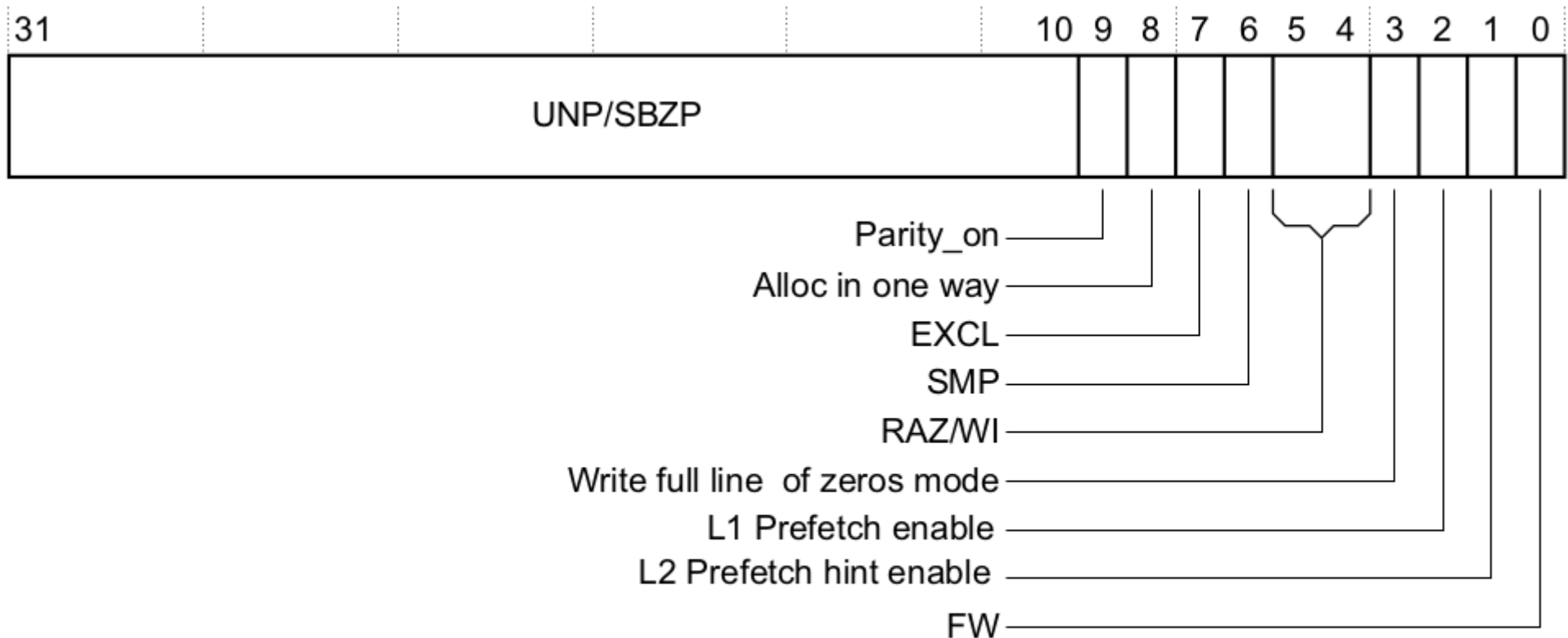


Figure 4-9 ACTLR bit assignments

L1 Prefetch enable – разрешаване на предсказването на преходи в кеш памет на ниво L1. 0 – забранено, 1 - разрешено

L2 Prefetch hint enable – разрешаване на предсказването на преходи в кеш памет на ниво L2.

Динамично предсказване на преходите

Cortex-A9 има **динамично** предсказване на преходите на **2 нива**, което включва [3]:

- ***ГНВ** (Global History Buffer == ВНТ) – 4096 2-битови предсказващи елемента

- ***ВТАС** (Branch Target Address Cache) – RAM памет с 2x256 клетки за буфериране на адреси на прехода

- ***RS** (Return Address Stack) – 8 32-битови думи за адреси на връщане

- *модулът е част от кеш L1

Динамично прелсказване на прехолите

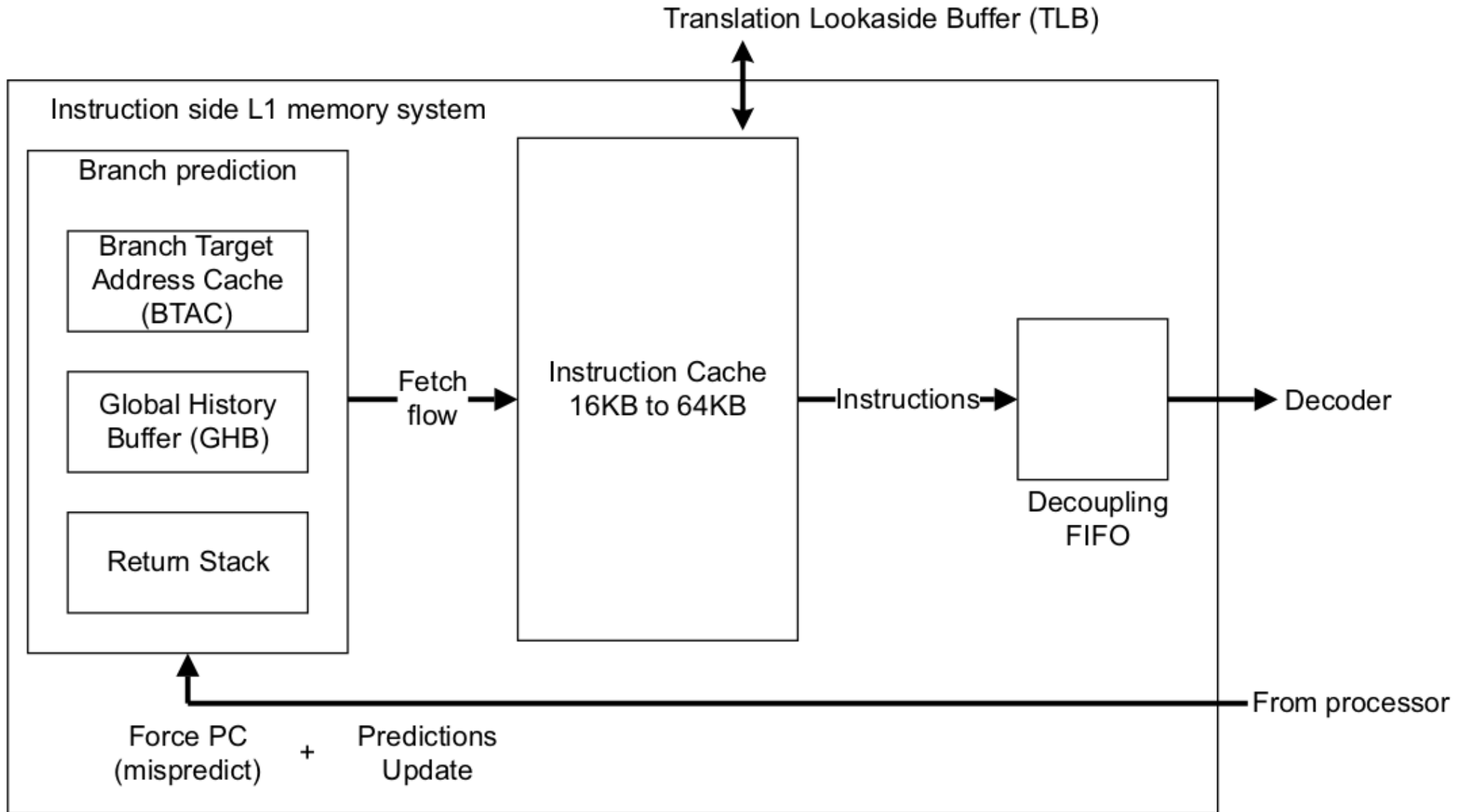


Figure 7-1 Branch prediction and instruction cache

Структурна схема на предсказване на преходите в ARM Cortex-A9 процесор [3].

Динамично предсказване на преходите

Предсказването на преходите включва следните инструкции:

- *условни (BNE, BEQ) и безусловни (B, BL) преходи**
- *безусловни с промяна набора инструкции (BLX)**
ARM→Thumb и Thumb → ARM
- *които имат за адрес на прехода фиксирано отместване спрямо PC;**
- *зареждания (LDR) и копирания (MOV) в PC,**
които най-вероятно са връщане от подпрограма. Това включва:
 - зареждания на PC от R13 (стековия указател)
 - зареждания на PC от R14 (линк регистър)

Динамично предсказване на преходите

Следните инструкции не могат да бъдат предсказвани:

- ***безусловни с промяна** набора инструкции (ARM, Thumb, ThumbEE, Jazelle), с изключение на ARM+Thumb комбинацията

- ***инструкции със суфикс S**

- ***инструкции за промяна режима на работа** (привилегирован/непривилегирован)

Динамично предсказване на преходите

RS предсказания се използват за следните инструкции (+режими на адресация):

PUSH в RS

- *BL непосредствена
- *BLX непосредствена
- *BLX регистрова
- *HBL (ThumbEE инструкция)
- *HBLP (ThumbEE инструкция)

POP от RS

- *BX R14
- *MOV PC, R14
- *LDM R13, {....., PC}
- *LDR PC, [R13]

Динамично предсказване на преходите

RS предсказания **не важат** за следните инструкции:

- *LDM с регистри, различни от R13 и PC

- *MOVS PC, R14

Тези инструкции се използват за връщане от хендлери на прекъсванията и променят режима на работа на ядрото (привилегирован / непривилегирован).

Литература

- [1] “Advanced Pipelining: Branch Prediction, Exceptions, and Limits to Pipelining”, ARM Education kit, Module 4, ARM Ltd, 2021.
- [2] “ARM1156T2-S Technical Reference Manual”, процесор ARM1156T2-S, ядро ARM11, набор инструкции ARMv6, r0p4, DDI 0338G, ARM Ltd, 2007.
- [3] “Cortex-A9 Technical Reference Manual”, r3p0, DDI 0388G (ID072711), ARM Ltd, 2011.
- [4] “The ARM Cortex-A9 Processors”, White Paper v2.0, ARM Ltd, 2009.
- [5] “ARM Cortex-A Series Programmer’s Guide”, Version 4.0, ARM DEN0013D (ID012214), 2013.
- [6] M. Lancaster, “CSCI 6461: Computer Architecture Branch Prediction”, lecture presentations, slide 18, The George Washington University, online, 2012.
- [7] N. Gloy, M. Smith, C. Young, “Performance Issues in Correlated Branch Prediction Schemes”, MICRO 28: Proceedings of the 28th annual international symposium on Microarchitecture, pp.3-14, 1995.

Външни връзки

- [a] <https://github.com/arm-university/Introduction-to-Computer-Architecture-Education-Kit>
- [b] <https://www.cs.umb.edu/cs641/notes26.html>
- [c] <https://www.cs.umd.edu/~meesh/411/CA-online/chapter/dynamic-branch-prediction/index.html>