Computer architecture

Napredni pipeline: Predviđanje grananja, iznimke i ograničenja u pipelining konceptu

Teme modula:

- Smanjivanje utjecaja grananja (upravljački hazardi)
- Upravljanje iznimkama
- Ograničenja u pipeline modelu

Upravljački hazardi

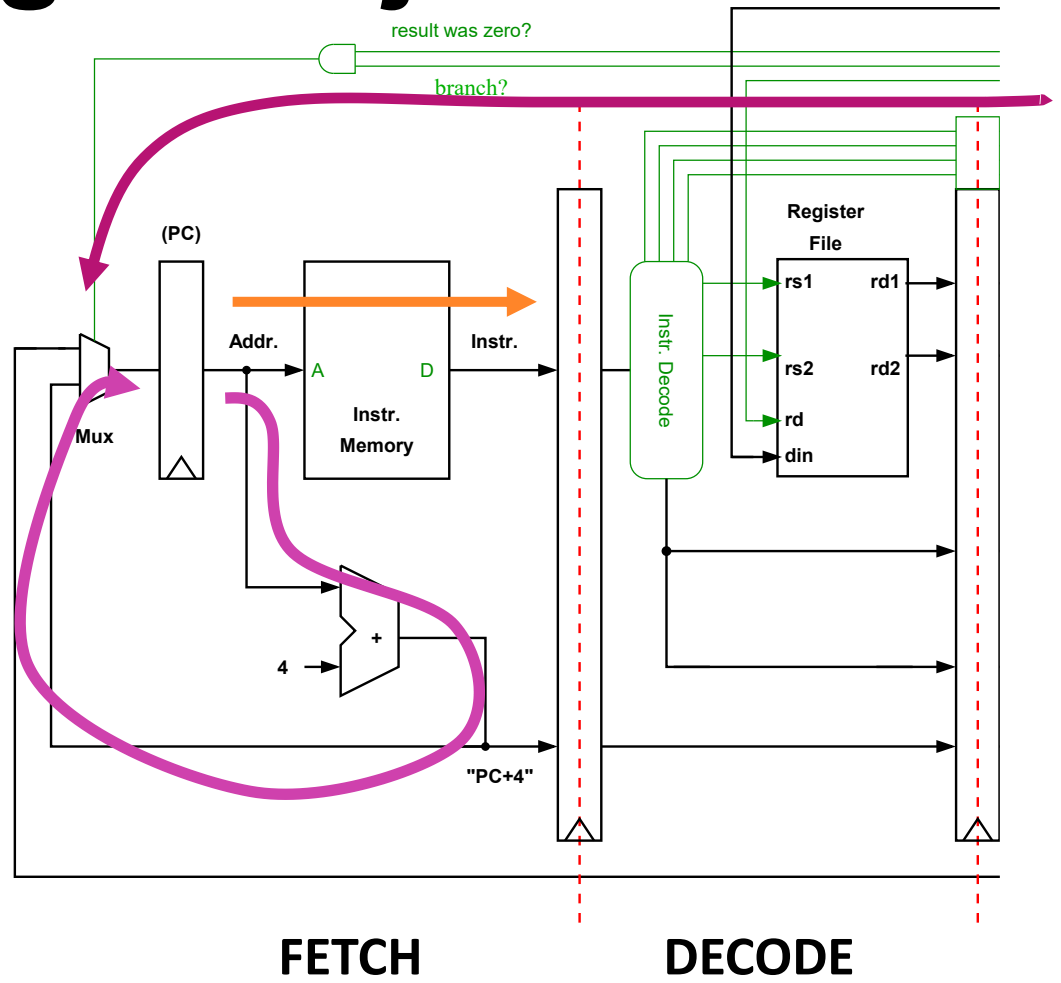
Što se događa neposredno nakon što dohvatimo (fetch) instrukciju sa uvjetnim grananjem?

1. Moramo odlučiti događa li se grananje ili ne
2. Ako se grananje događa moramo izračunati i prebaciti odredišnu adresu.

Kako želimo realizirati grananje?

U procesoru koji sadrži pipelineove, željeli bismo izračunati sljedeću vrijednost brojača programa (PC) paralelno s čitanjem naše memorije s instrukcijama.

U stvarnosti ovo je netrivijalno jer se grananja mogu računati kasnije u procesu pipelinea te se rezultat njihovog grananja ne zna do tog trenutka.



Kako bismo željeli rukovati grananjem?

- Opcija 1: Pretpostavimo da se grananje nije dogodilo
- Opcija 2: Izračunamo grananje što je ranije moguće
- Opcija 3: Pričekamo na grananje
- Opcija 4: Predviđanje grananja

Pretpostavimo da se grananje nije dogodilo

(CBZ – grananje ako je rezultat jednak nuli)

Ako se grananje ocjenjuje u fazi izvršenja, a izvršava se, moramo pretvoriti dvije upute koje je slijede u NOP-ove (gubimo dva ciklusa).

Pretpostavimo da se grananje nije dogodilo

Ako procijenimo granu u fazi izvršenja, izgubili bismo dva ciklusa svaki put kad bismo naišli na grananje koje trebamo izvršiti.

Ako pretpostavimo da su 20% instrukcija grananja, i 60% moramo izvršiti, a također pretpostavimo i da je CPI 1:

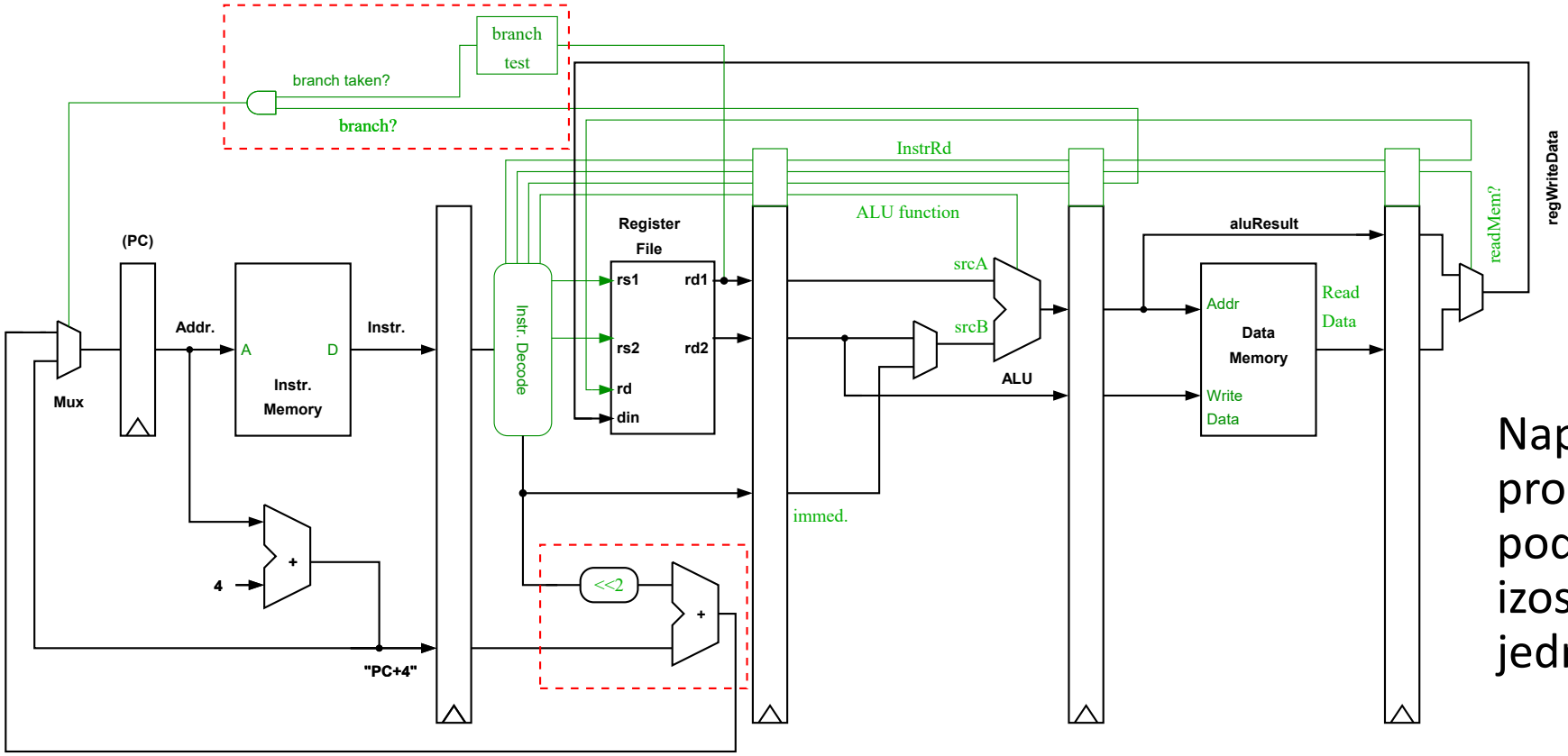
CPI od grananja koja su se dogodila= 0.2*2 * 0.6 = **0.24**

Novi CPI = 1.0 + kašnjenje u grananjima = 1.24

Izračunamo grananje što je ranije moguće

- Premjestite test grananja i izračun ciljne adrese grananja u fazu dekodiranja.
- Time bi se kazna grananja smanjila na jedan ciklus u slučaju izvršenog grananja.
  - i.e., U slučaju izvršenog grananja moramo odbaciti iduću instrukciju u pipelineu

Izračunamo grananje što je ranije moguće



Napomena: Logika prosljeđivanja podataka izostavljena radi jednostavnosti

Izračunamo grananje što je ranije moguće

Da bi ova tehnika djelovala:

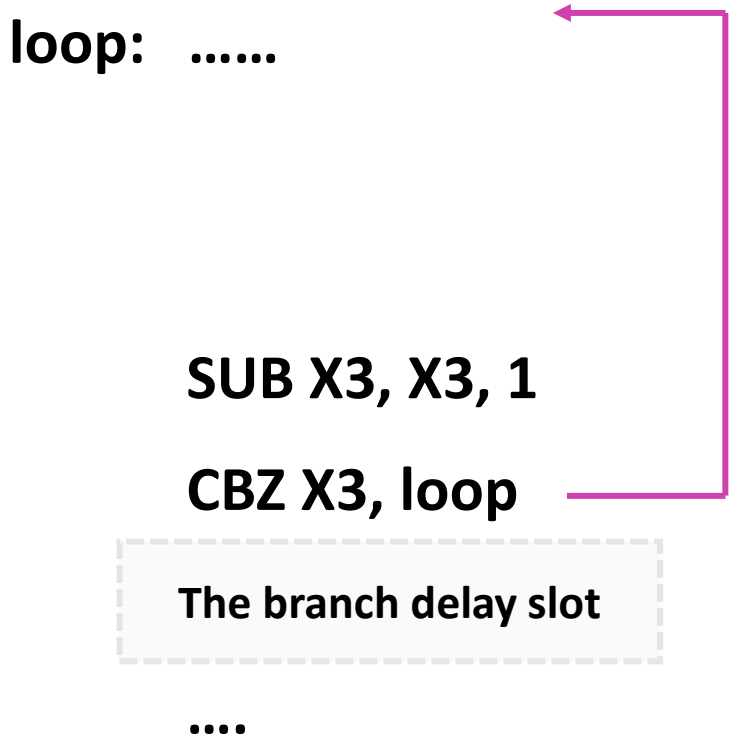
Stanje grananja mora biti jednostavno procijeniti.

- Test za nulu je jednostavan .
- Testovi koji zahtijevaju ALU operaciju vjerojatno su previše složeni.
- Ne želimo da test grananja produži vrijeme ciklusa.

- Moramo se pobrinuti za potencijalne podatkovne rizike.
  - Ako neposredno neposredno prije grananja piše u registar koji se koristi za testiranje uvjeta grananja, moramo odugovlačiti jedan ciklus (tj. dok ova instrukcija ne generira svoj rezultat).
  - Trebat će nam i putevi prosljeđivanja od faza EXE i MEM do faze dekodiranja.

•Pričekamo na grananje

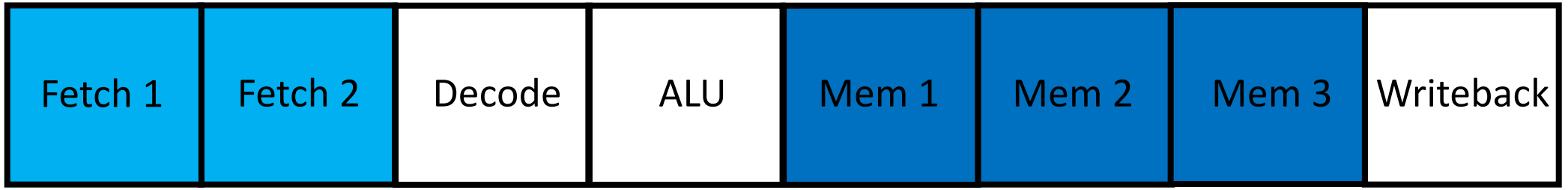
- Mogli bismo odlučiti uvijek izvršiti instrukcije nakon grananja, bez obzira na to je li grananje izvršeno
- Ova instrukcija iza grananja se sad zove **“branch delay slot.”**



Pričekamo na grananje

- Kompajler obično može popuniti branch delay slot 60-70% ukupnog vremena izvršavanja.
- Ako imamo više faza u pipelineu možemo kreirati dodatne delay slotove no time ih ima previše pa ih je teško popuniti.
- Pitanje: gdje možemo pronaći instrukciju kojom možemo popuniti delay slot?
- Ako je instrukciju nemoguće pronaći onda na njezino mjesto ide NOP instrukcija.

Primjer



Ovaj pipeline ima 8 faza

- Osnovno kašnjenje zbog grananja je 3 ciklusa.
- Uvjet grananja i odredišna adresa se računaju u fazi ALU.
- Sva grananja smatramo kao da neće biti izvršena.

Primjer

- Kako to utječe na CPI:
  - Napravimo neke razumne pretpostavke:
  - 15% instrukcija su bezuvjetne ili izvršena grananja; posljedica im je kašnjenje 3 ciklusa.
  - 5% instrukcija su neizvršena grananja. Kako pretpostavljamo da je to svaki put slučaj ovo ne stvara kašnjenje
  - **Ukupni utjecaj na CPI zbog kašnjenja** = 0.15 * 3 + 0.05 * 0 = **0.45**

Predviđanje grananja

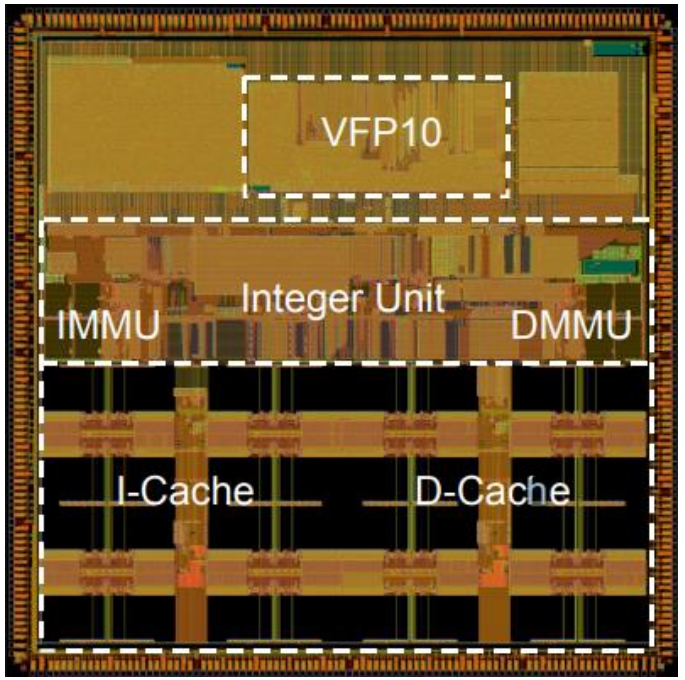
- Za procesore visokih performansi, s dubokim pipeline strukturama, do sada opisane tehnike nisu dovoljne.
- Primjerice **Arm Cortex-A15**
  - 15 stupnjeva pipelinea
  - Grananja se računaju u kasnijim fazama izvršavanja.
  - Svaka greška u procjeni grananja je ~14 ciklusa kašnjenja
  - Procesor dohvaća do 4 instrukcije po ciklusu i dekodira po 3 istovremeno
  - Rezultat svega ovoga je da krivo procijenjeno grananje može značiti ponovno izvršavanje više od 40 instrukcija.

Statičko predviđanje grananja

- Metode statičkog predviđanja iskorištavaju opažanje da će određeno grananje vjerojatno biti vrlo pristrano u jednom smjeru.
- Sheme se često temelje na tome grana li se grana naprijed ili natrag u kodu ili alternativno ovise o op-kodu instrukcija u grananju.
- **Predviđanje temeljeno na pomaku**
  - Možemo iskoristiti jednostavno opažanje da su grane koje se granaju unazad (prema nižoj adresi) obično grane petlje i da će vjerojatno biti aktivirane .
  - Ako ciljna adresa grananja < PC, predviđamo da će grananje biti izvršeno.
  - Točnost ove vrste sheme je oko 65% (ili 80-95% uz pomoć profiliranja).

Primjer: Procesor Arm10

- Cijeli pipeline imao je 6 faza.
- Arm10 je koristio shemu predviđanja grana temeljenu na statičkom predviđanju.
- Dio za predviđanje grananja u Arm10 također je radio prije faze dohvaćanja (upute za prefetching) kako bi se smanjili troškovi grananja.



Arm10 (Circa 2000)

Transistor count: ~250,000

Clock frequency: 400 MHz

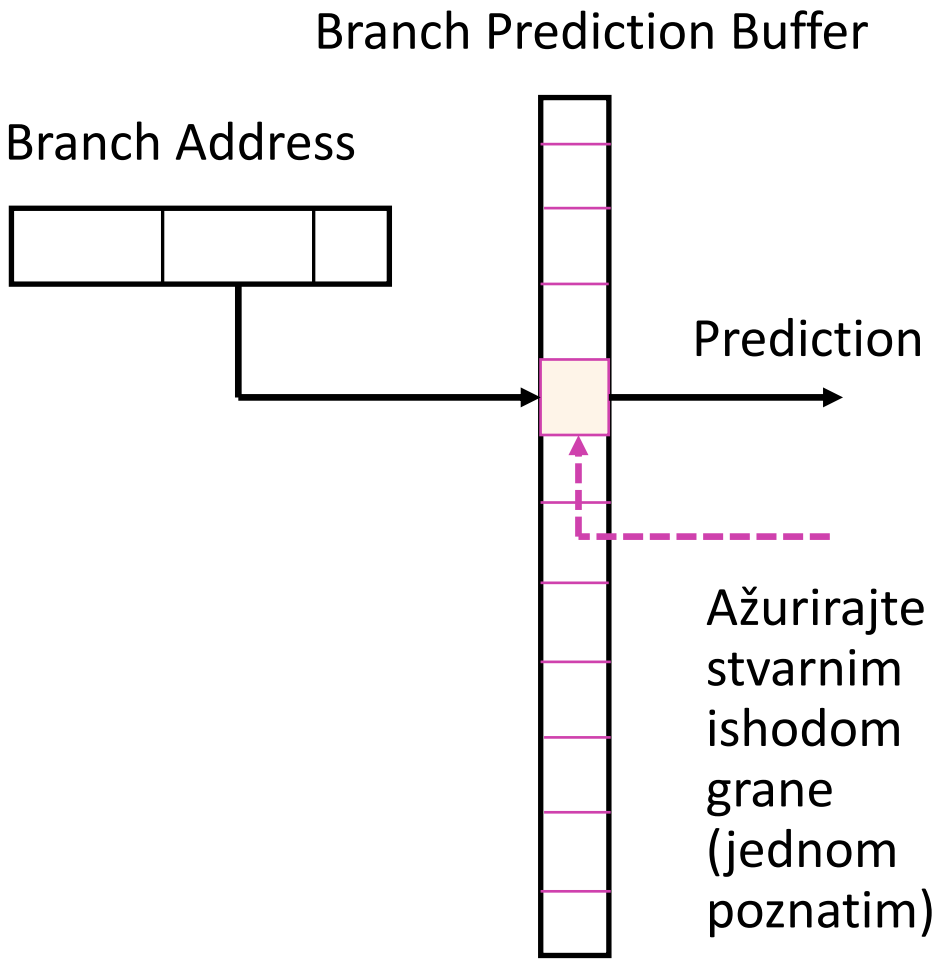
Technology: 0.18-micron

Applications: modems, cellular phones, automotive, etc.

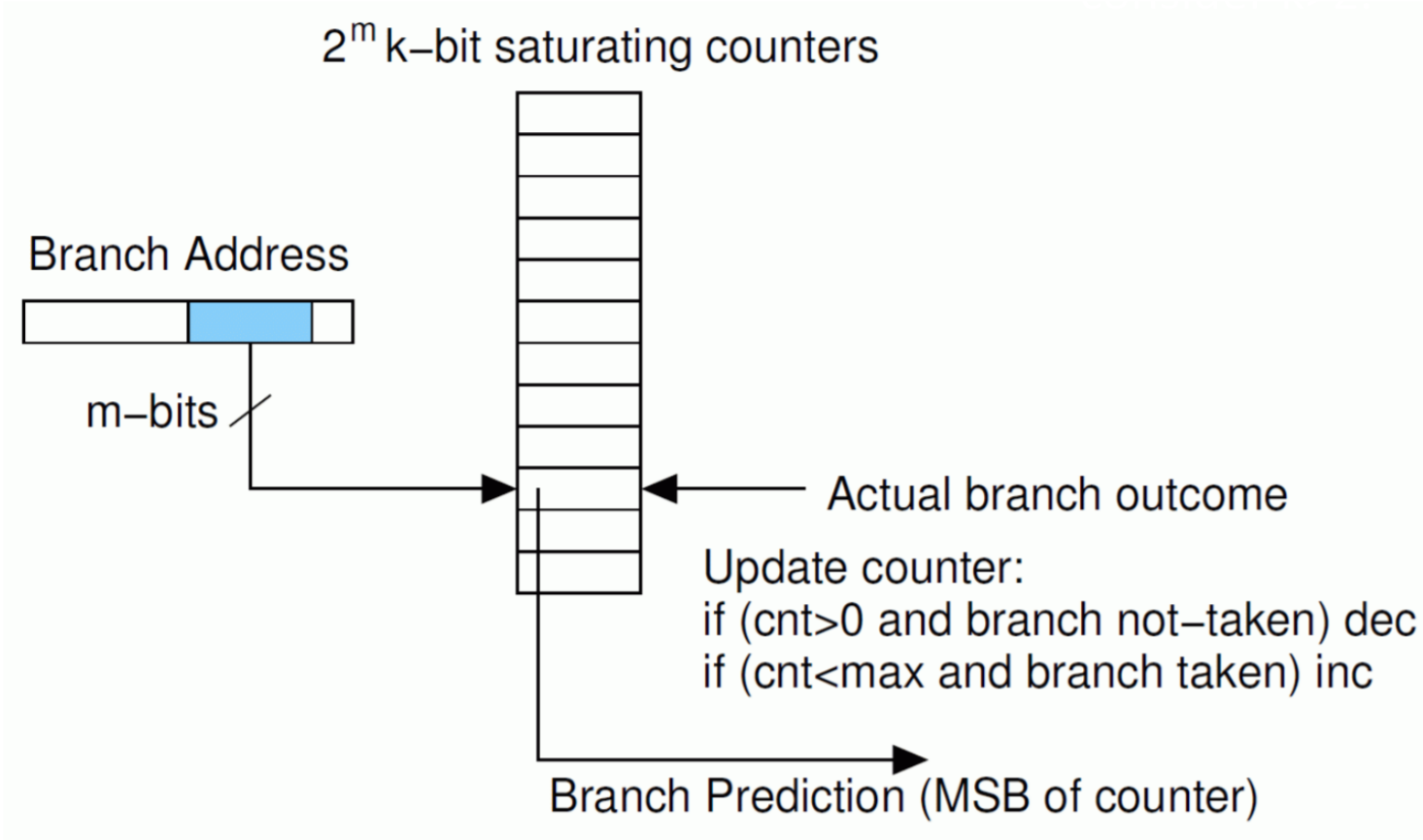
Jednostavan dinamički prediktor grananja na jednoj razini

Jednostavan algoritam za predviđanje dinamičkog grananja će predvidjeti granu kao izvršenu samo ako je to bilo tako posljednji put kad smo je izvršili. Možemo koristiti jednostavnu tablicu jednobitnih unosa za pohranu naših predviđanja.

Nedostatak ovih jednobitnih unosa u tablici je u tome što jedan događaj može preokrenuti predviđanje. Vjerojatno želimo neku histerezu, npr. u slučaju petlji kako bismo izbjegli dva pogrešna predviđanja po petlji (pri ulasku i izlasku).



One-level Branch Predictors



Korelirajući prediktori

- Kako možemo poboljšati jednostavne bimodalne prediktore?
- Možemo iskoristiti činjenicu da je ishod mnogih grana povezan ili s prošlim ishodima iste grane (lokalna povijest) ili s drugim novijim granama (globalna povijest).
- Sada umjesto jednostavnog mijenjanja PC baziranog na hashingu kako bismo odabrali brojač, možemo uključiti lokalnu ili globalnu povijest podružnica kako bismo poboljšali naše predviđanje.

Korelirajući prediktori (lokalna povijest)

Ako ishod određene grane ima ponavljajući obrazac, njezina lokalna povijest može se koristiti za poboljšanje točnosti predviđanja.

E.g., određena instrukcija može biti predvidljiva ako pogledamo njezinu lokalnu povijest:

….01011010110101101011010110101 1

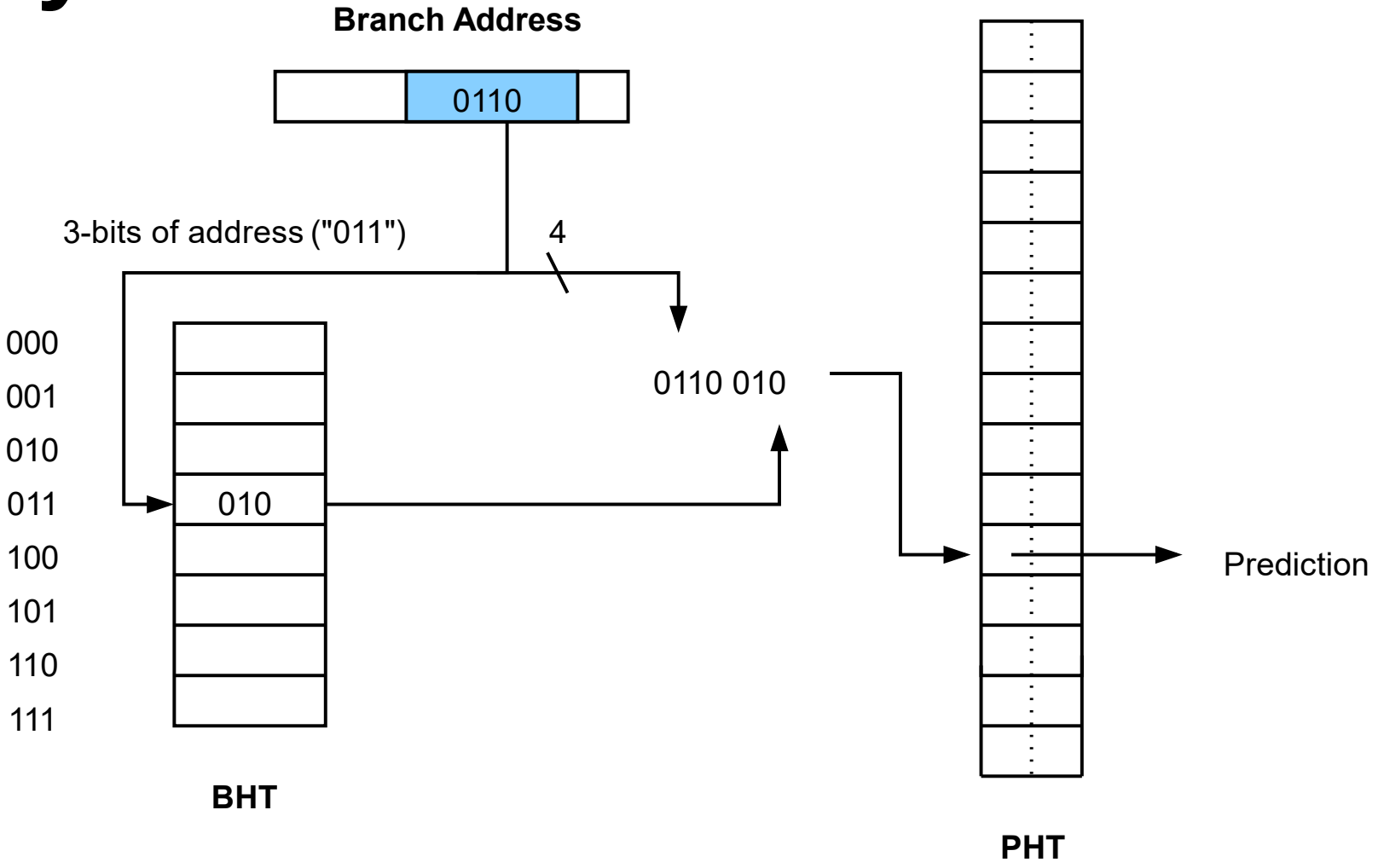
History	Next
0101	1
1011	0
0110	1
…	

Local-history Two-level Predictor

Sada imamo dvije memorije ili tablice:

The Branch History Table (BHT)

The Pattern History Table (PHT)

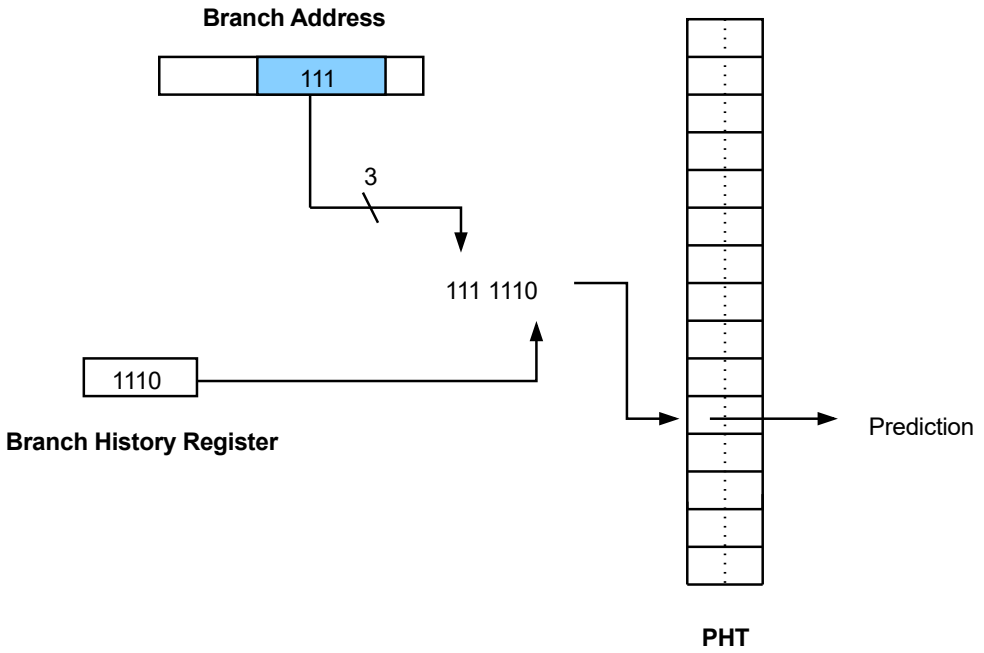


Iskorištavanje povijesti globalnih grana

- Osim iskorištavanja lokalne povijesti određene grane, također možemo primijetiti da je ponašanje grane često povezano s ponašanjem drugih novijih grana (globalna povijest):
-
- `If (cond1) { .... }`
- `If (cond2) { .... }`
- `If (cond1 && cond2) { .... }  // dependent on outcome of`

  `// previous branches`

Global History Two-level Predictor



Optimizacije

Tournament Prediktori

Naprednije sheme koriste lokalne i globalne prediktore povijesti

Ti takozvani "prediktori turnira" također koriste treći prediktor (još jednu tablicu dvobitnih brojača) kako bi odabrali koji će izlaz prediktora (lokalno ili globalno) koristiti.

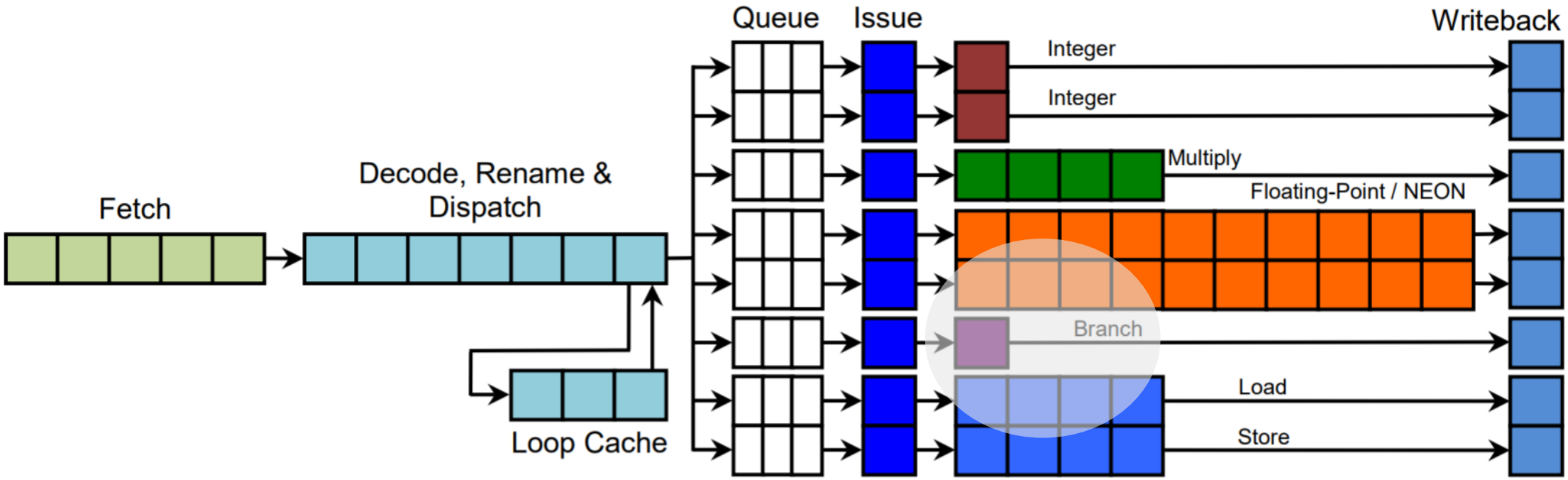
Problemi s pseudonimom

Performanse mogu biti ograničene negativnim smetnjama, odnosno kada se dvije grane mapiraju na isti unos u PHT, ali su pristrane u suprotnim smjerovima. Brojne sheme pokušavaju se nositi s tim, npr. korištenjem više PHT-ova ili korištenjem male označene "predmemorije" za držanje grana koje doživljavaju smetnje.

Ograničenja dinamičkog predviđanja grane

- Neke grane su nepredvidive (ovise o ulaznim podacima).
- Trebate "trenirati" prediktor za neko razdoblje prije nego što predviđanja budu točna
- Točnost prediktora bit će ograničena područjem (trošak), vremenom ciklusa ili snagom hardvera.
- Pseudonim i smetnje

Primjer: Arm Cortex-A15



Arm Cortex-A15 pipeline

Primjer: Arm Cortex-A15

Cortex A-15 koristi prediktor "dvosmjernog načina rada", produžetak globalnog prediktora na dvije razine povijesti kako bi se smanjila ograničenja zbog destruktivnog pseudonima.

Sada postoje dva PHT-a, svaki s 8192 unosa i dodatnim "izbornim" prediktorom za odabir između dva PHT-a.

Sveukupno, prediktor grane je relativno velik i troši ~15% snage jezgre.

Grananje bez zastoja

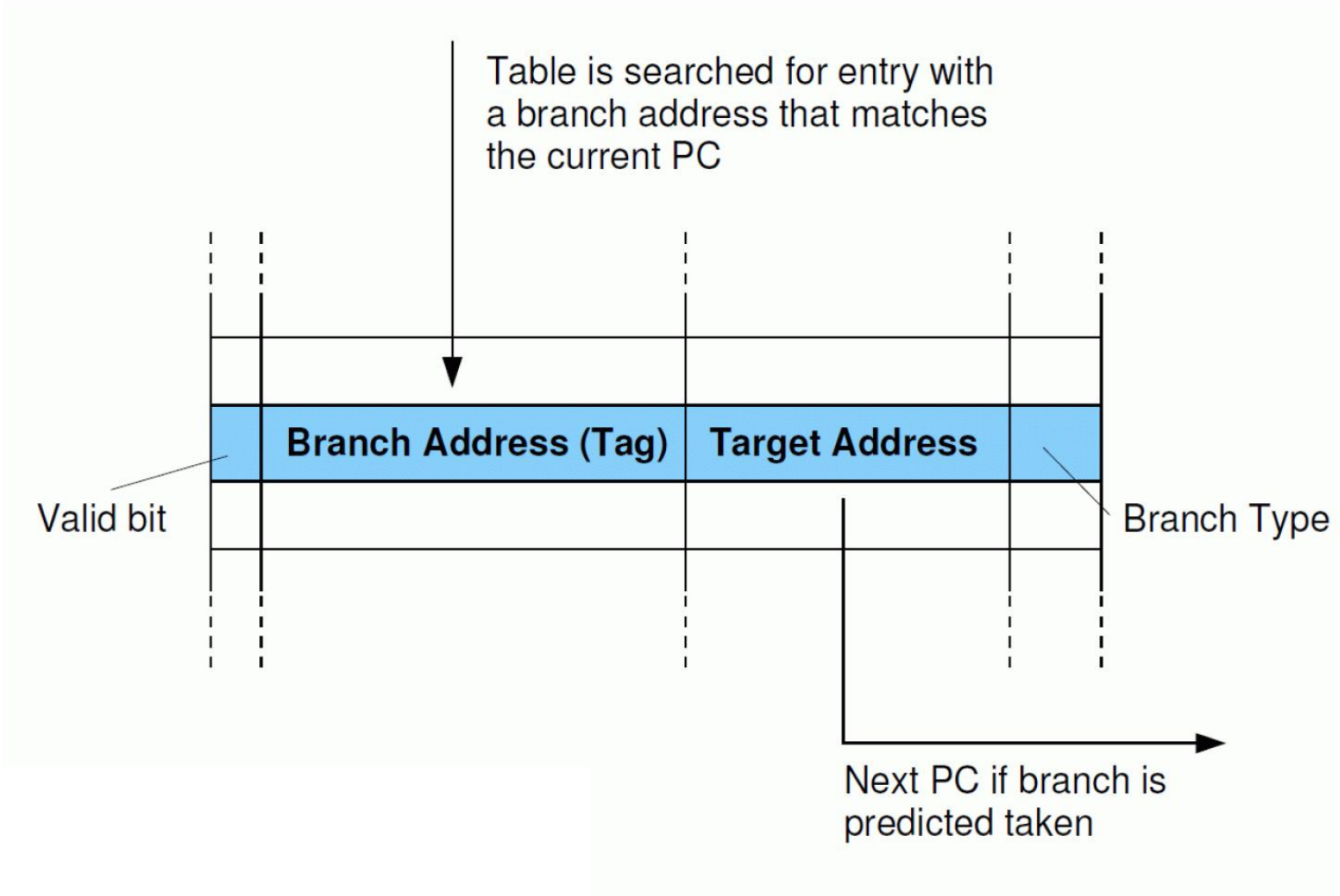
Što trebamo znati kako bismo u potpunosti izbjegli usporavanja u grananju?

Trebate znati koja instrukcija koje "trenutno" donosimo je grananje (zapamtite, još se nije vratila iz memorije, pa kako možemo znati?)

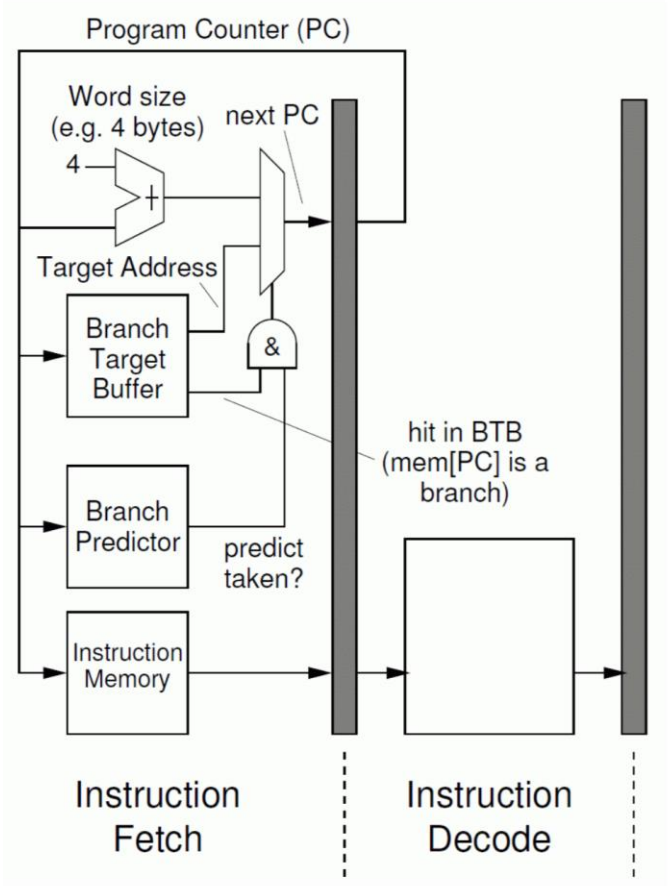
1. Moramo predvidjeti hoće li se grananje dogoditi ili ne
2. Ako se grananje događa moramo odrediti odredišnu adresu.

Da bismo pružili informacije za rješavanje problema 1 i 3, nedavna grananja pohranjujemo zajedno s njihovim ciljnim adresama u Bransch target bufferu (BTB).

The Branch Target Buffer (BTB)



Kompletan sustav



Rest of pipeline…

Ostali BTB trikovi: Preklapanje grane

- Osim pohrane ciljne adrese, ciljne instrukcije mogli bismo pohraniti u BTB.
- Nema potrebe za dohvaćanjem sljedećih uputa; CPI za grananje je sada zapravo nula.
- Također bi mogli omogućiti da odvojimo više vremena za pristup (većem) BTB
- Grananje je sada uklonjeno iz slijeda instrukcija koji je predstavljen pipelineu za izvršenje - umjesto toga, grana se zamjenjuje ciljnom instrukcijom grane
- Također možemo stvoriti zasebnu strukturu za predmemoriranje instrukcija za cilj grananja (i.e. the **Branch Target Instruction Cache** or **BTIC**)

Prediktori povratnih adresa

Funkcije se mogu pozvati s više mjesta u programu.

Točnost cilja grane (povratne adrese) pohranjene u BTB-u može biti vrlo niska.

Rješenje: za pohranu tih adresa koristite mali hardverski stog (stog povratne adrese).

Što ako se ovaj stog prelijeva?

Iznimke

Iznimke

- U nekim situacijama moramo prekinuti izvršenje programa i poduzeti neke radnje. Ti se uvjeti ili događaji sustava nazivaju iznimkama. Potrebne radnje poduzima povlašteni softver, takozvani *exception handler* ili softver za obradu iznimaka.
- Iznimke se mogu pojaviti iz više različitih razloga,
  - page fault, breakpoint, I/O device request, floating-point errors, memory protection violation

Iznimke

Vrste iznimaka (za Arm):

- Interrupts
- Aborts
- Reset
- Exception generating instructions

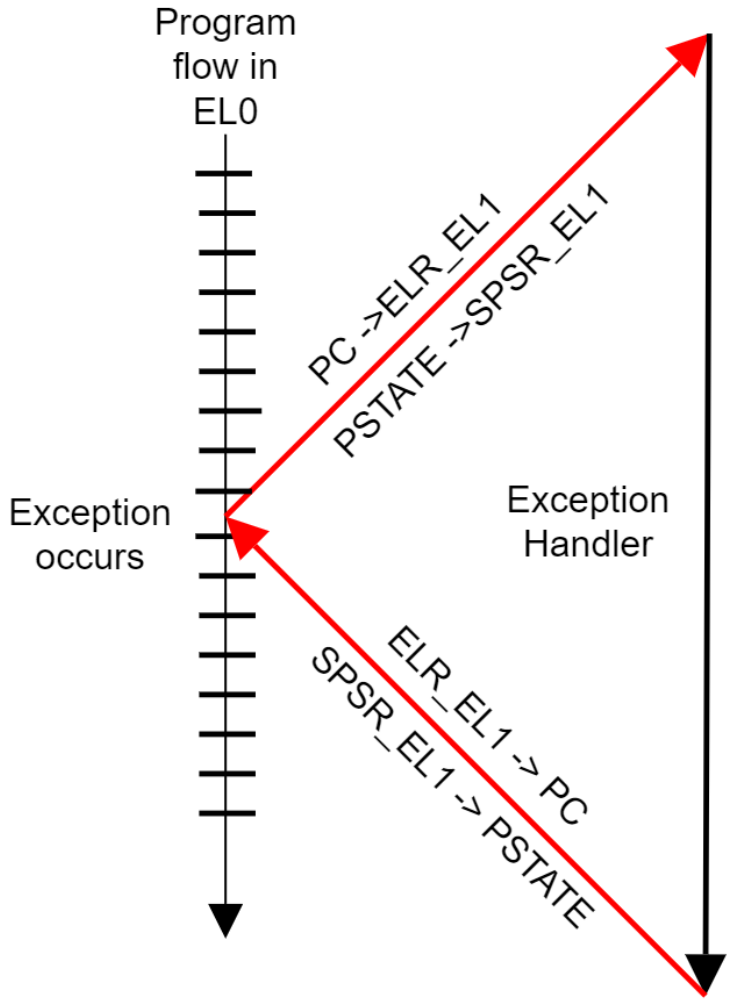
Iznimke

Iznimka će uzrokovati da procesor izvrši sljedeće:

Spremite stanje procesora (PSTATE), npr. zastavice procesora, bitove maske prekida, razinu iznimke itd.

Spremanje povratne adrese (trenutni PC).

1. Skok na dio za rukovanje iznimkom definiran vektorom u memoriji
2. Spremite registre, izvršite kod rukovatelja iznimke, vratite registre.
3. Povratak iz iznimke (instrukcija ERET).



Iznimke

Namjera je privremeno prekinuti izvršavanje programa, riješiti iznimku, a zatim nastaviti izvršavanje.

Dobar način da osiguramo da možemo lako nastaviti je osigurati da stanje procesora bude u skladu sa sekvencijalnim modelom izvođenja programa prije nego što se izvrši iznimka, odnosno:

Sve instrukcije prije E trebale su ispuniti i ažurirati svoje odredišne registre. Trebalo je postupati sa svim iznimkama uzrokovanim ovim instrukcijama.

Sve instrukcije nakon E u programskom redoslijedu ne bi trebale biti dovršene i nisu trebale mijenjati nijedno stanje procesora.

Hoće li se E dovršiti ili ne, ovisit će o iznimki.

To se naziva "preciznim iznimkama".

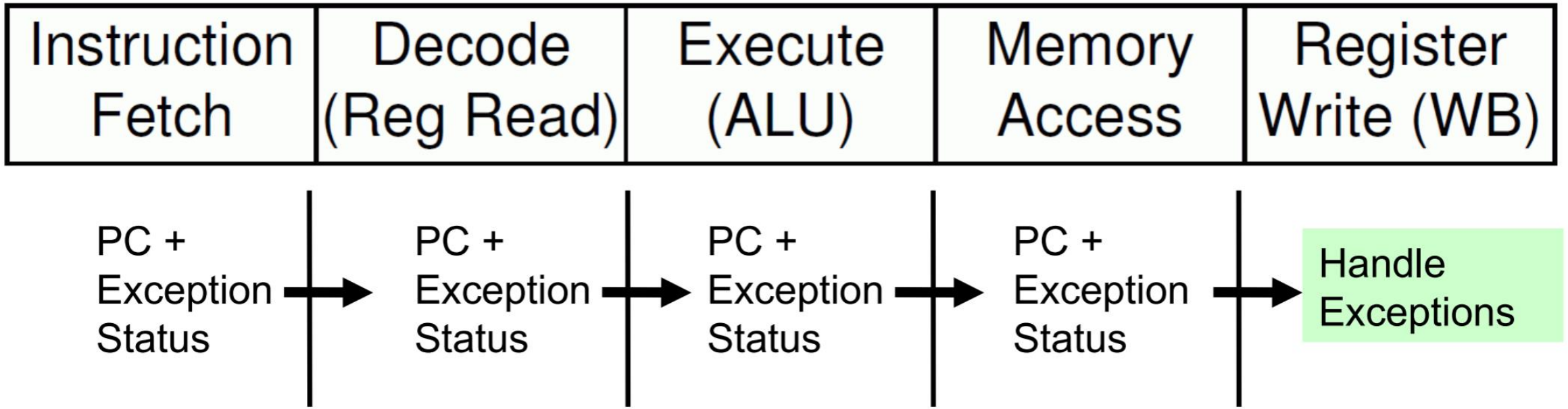
Precizne iznimke i pipelining

Zahtjevi na prethodnom slajdu su trivijalni u slučaju nepipeliranog procesora, ali složeniji za procesor sa pipelineom.

V, U, T, S, R, Q, P, O, N, M, L, K, J, I, ….

V, U, T, S, R, Q, P, O, N, M, L, K, J, I, ….

Precizne iznimke



Granice pipelininga

Ograničenja pipelininga

- Kao što smo vidjeli u posljednjem modulu, dublji pipeline ne mora nužno dovesti do boljih performansi.
- Moramo naporno raditi kako bismo puniti pipeline i podacima i instrukcijama kako bi minimizirali zastoje
- Ako unutar pipelinea imamo i našu fazu izvršenja, također ćemo morati pronaći uzastopne neovisne instrukcije kako ne bi imali puno čekanja

Ograničenja pipelininga

Pipelining je u konačnici ograničen i poteškoćama na nižoj razini:

Registarski i troškovi izvršenja ciklusa nisu nula. Ako imamo vrlo malo logike po fazi pipelinea, to može predstavljati značajan dio našeg puta kritičnog kašnjenja.

Trebate uravnotežiti logiku između faza. Vremensko razdoblje ciklusa određeno je kašnjenjem u najgorem slučaju.

Ograničenja broja registara u pipelineu

Ograničenja pipelininga

- U konačnici, nećemo moći povećati performanse procesora koji pokušava izdati samo jednu uputu po ciklusu.
- Čak i prije nego što se dostigne ova točka, poželjno je koristiti druge tehnike: obavljati više posla u svakoj fazi, dohvatiti i izvršiti više instrukcija po ciklusu.

Hvala vam na pažnji!