Транслација инструкција

Садржај

- *CISC* и *RISC*
- Претварање *CISC* у *RISC*
- Операције
  - Микроперације
  - Макрооперације
- Оптимизације
  - Кеширање
  - Спајање
- Примери процесора

CISC и *RISC*

- *RISC*: *reduced-instruction set computer*
  - Термин настао почетком осамдесетих (*Patterson*)
  - *RISC-I* (*Patterson*), *MIPS* (*Hennessy*), *IBM 801* (*Cocke*)
  - Примери: *PowerPC*, *ARM*, *SPARC*, *Alpha*, *PA-RISC*

- *CISC*: *complex-instruction set computer*
  - Овај термин није постојао пре термина *RISC*
  - Примери: *x86*, *VAX*, *Motorola 68000*, ...

- Концептуална борба започела средином осамдесетих
  - *RISC* приступ добио технолошке бике
  - *CISC* приступ добио на полу напредних решења (од деведесетих до сада)
  - *RISC* приступ добио на полу мобилних и уграђених уређаја

Пример *CISC*

- *VAX* (***V**irtual **A**ddress e**X**tension to PDP-11*)
  - Променљива дужина инструкције: 1-321 бајта!!!
  - 14 регистара + PC + SP + бити услова
  - Ширина података: 8, 16, 32, 64, 128 бита, децимални бројеви, стрингови
  - Инструкције за приступ меморији за све типове података
  - Специјалне инструкције: *crc*, *insque*, *polyf*, као и велики број конверзија

Пример *CISC*

- *x86*: „*Difficult to explain and impossible to love*”
  - Променљива дужина инструкције : 1-16 бајтова
  - 8 специјализованих регистара + бити услова (преклопљени регистри)
  - Ширина података: 8, 16, 32, 64 бита
  - Акумулаторски рад (регистри и меморија) за целе бројеве, стековски рад за бројеве у покретном зарезу
  - Адресирања: индиректно, скалирано, са померајем, + сегменти, ...
  - Специјалне инструкције: *push, pop,* стринг инструкције, *MMX*, *SSE /2/3*

x86 – ADD инстукција

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
04 ib	ADD AL, imm8	I	Valid	Valid	Add imm8 to AL.
05 iw	ADD AX, imm16	I	Valid	Valid	Add imm16 to AX.
05 id	ADD EAX, imm32	I	Valid	Valid	Add imm32 to EAX.
REX.W+05 id	ADD RAX, imm32	I	Valid	N.E.	Add imm32 sign-extended to 64-bits to RAX.
80 /0 ib	ADD r/m8, imm8	MI	Valid	Valid	Add imm8 to r/m8.
REX+80 /0 ib	ADD r/m8*, imm8	MI	Valid	N.E.	Add sign-extended imm8 to r/m8.
81 /0 iw	ADD r/m16, imm16	MI	Valid	Valid	Add imm16 to r/m16.
81 /0 id	ADD r/m32, imm32	MI	Valid	Valid	Add imm32 to r/m32.
REX.W+81 /0 id	ADD r/m64, imm32	MI	Valid	N.E.	Add imm32 sign-extended to 64-bits to r/m64.
83 /0 ib	ADD r/m16, imm8	MI	Valid	Valid	Add sign-extended imm8 to r/m16.
83 /0 ib	ADD r/m32, imm8	MI	Valid	Valid	Add sign-extended imm8 to r/m32.
REX.W+83 /0 ib	ADD r/m64, imm8	MI	Valid	N.E.	Add sign-extended imm8 to r/m64.
00 /r	ADD r/m8, r8	MR	Valid	Valid	Add r8 to r/m8.
REX+00 /r	ADD r/m8*, r8*	MR	Valid	N.E.	Add r8 to r/m8.
01 /r	ADD r/m16, r16	MR	Valid	Valid	Add r16 to r/m16.
01 /r	ADD r/m32, r32	MR	Valid	Valid	Add r32 to r/m32.
REX.W+01 /r	ADD r/m64, r64	MR	Valid	N.E.	Add r64 to r/m64.
02 /r	ADD r8, r/m8	RM	Valid	Valid	Add r/m8 to r8.
REX+02 /r	ADD r8*, r/m8*	RM	Valid	N.E.	Add r/m8 to r8.
03 /r	ADD r16, r/m16	RM	Valid	Valid	Add r/m16 to r16.
03 /r	ADD r32, r/m32	RM	Valid	Valid	Add r/m32 to r32.
REX.W+03 /r	ADD r64, r/m64	RM	Valid	N.E.	Add r/m64 to r64.

*In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Пример *RISC*

- *MIPS, PA-RISC, SPARC, PowerPC, Alpha, ARM*
  - 32 битне инструкције
  - 32 регистра за рад са целобројним величинама
  - 32 регистра за рад са величинама у покретном зарезом
  - Load/store архитектура са мало адресирања
  - Пуно сличних архитектура, сваки произвођач за себе

Пример *RISC*

- *DEC Alpha* (*Extended VAX*):
  - Чист *RISC*
  - 64 битни подаци (32, 16, 8 додати касније)
  - Све адресу су поравнате
  - Једино меморијско адресирање регистарско индиректно са померајем
  - Специјалне инструкције: условна смештања, довлачење унапред

Историјски развој

- Пре 1980
  - Непоуздани преводиоци (ручно писање асемблера)
  - Комплексно, архитектура високог нивоа (**лако писање асемблера**)
  - Спора микропрограмске имплементација на више чипова
- Од 1982
  - Направљен процесор на чипу…
    - **…али само за једноставне архитектуре**
  - Перформансе овакве интеграције су биле очигледне
- **RISC манифест**: формирати архитектуру која …
  - **Поједноставује коришћење једног чипа**
  - **Олакшава оптимизације приликом превођења**

Скица архитектура

RISC	*CISC*
Извршавање траје један такт	Инструкције могу трајати више тактова
Ожичена реализација	Микропрограмска реализација за дуготрајне инструкције
Load/store архитектура	Регистар-меморија и меморија-меморија
Мало начина адресирања	Доста начина адресирања
Инструкције исте дужине	Много формата инструкција
Ослањање на оптимизације преводиоца	Ручно бирање инструкција даје добре перформансе
Много регистара (корисно за оптимизацију)	Мало регистара

Перформансе RISC наспрам CISC

- Једначина која описује перформансе:

  Перформансе ≈ 1 / Време извршавања

- Време извршавања = инструкције*тактова по инструкцији*период

- инструкције = L = инструкција/програм
- тактова по инструкцији = CPI = тактова/инструкцији
- период = Tc = 1 / f = секунди/тактова)

Перформансе RISC наспрам CISC

- *CISC*
  - Мали број инструкција по програму користећи комплексе инструкције
  - Повећање тактова по инструкцији или дужи период
  - Једноставан асемблер, велика густина кода
- RISC
  - Побољшан број тактова по инструкциј уз доста кратких инструкција
  - Повећан број инструкција по програму
    - Преводилац ово може смањити
  - Побољшан број тактова у јединици времена (секунди/тактова)
    - Коришћење једноставних инструкција

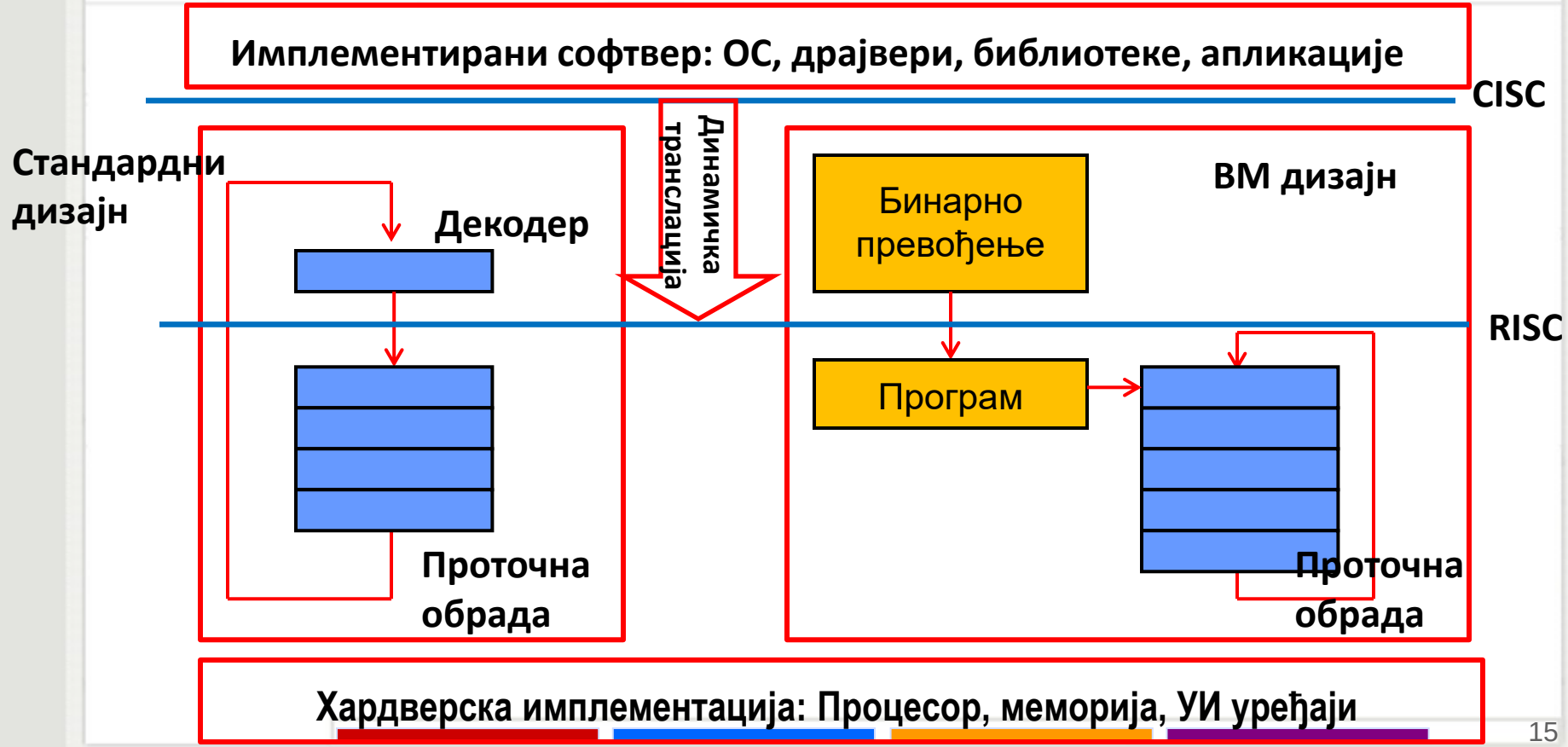
Поређење - дебата

- Аргументи на страни *RISC* архитектуре
  - CISC архитектура је у основи ограничена
  - За дату технологију RISC имплементације су боље/брже
    - Када је CISC захтева више чипова RISC је користио један чип
    - Када је CISC прешао на један чип RISC је добио проточну обраду
    - Када је CISC добио проточну обраду RISC је добио кеш
    - Када је CISC добио кеш RISC је добио више језгара
    - ...
- Аргументи на страни *CISC* архитектуре
  - CISC није ограничен само му треба **више транзистора**
  - Муров закон ће смањити разлику RISC/CISC разлику
    - Проточна обрада: RISC = 100K транзистора, CISC = 300K
    - Од 1995: 2M+ су изравнали разлике
  - Цена програма је доминантна, компатибилност је најважнија

Интелов x86 трик: RISC изнутра

- 1993: Интел је желео да направи извршавање ван редоследа (*out-of-order execution*) код Пентиума Про.
  Тешко са крупним инструкција као код x86 архитектуре
- Решење?
- Транслација из x86 у RISC микрооперације у хардверу!!

Интелов x86 трик: RISC изнутра



Интелов x86 трик: RISC изнутра

- Процесор **екстерно** задржава **x86** архитектуру због компатибилности
- Процесор **интерно** извршава **RISC mISA** због имплементације
- Коришћење превођења омогућава да се x86 извршава као да је RISC
  - Интел је имплементирао извршавање ван редоследа пре неке RISC компаније
  - Извршавање ван редоследа такође помаже x86 (зато што архитектура ограничава преводиоце)
- Ово користе и остале x86 имплементације (АМД)
- Различите микрооперације за различите дизајне рачунара
  - **Ово није део архитектуре!**
  - **Није јавно доступан**

Могући начини превођења

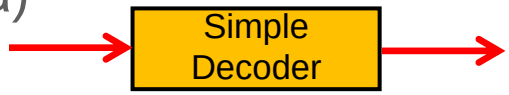
- Већина x86 инструкција се може пресликати на једну до три микрооперације
- Већина инструкција се пресликава у **једну** микрооперацију
  - add, xor, compare, branch, ...
  - Пример за Load: **mov -4(eax), ebx**
  - Пример за Store: **mov ebx, -4(rax)**
- Сваки приступ меморији додаје по микрооперацију
  - **addl -4(eax), ebx -** две микрооперације (load, add)
  - **addl ebx, -4(eax) -** три микрооперације (load, add, store)

Могући начини превођења

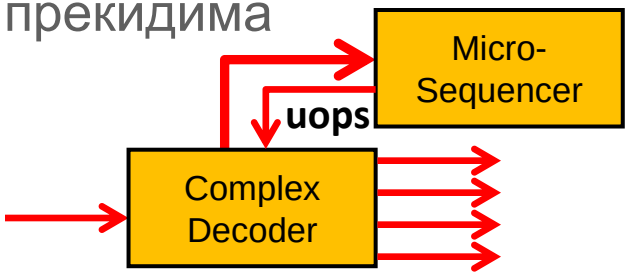
- Рад са стеком (PUSH/POP) – две микроперације
  - Приступи стеку, ажурирај показивач
    push eax
    Се може реализовати као:
    store eax, -4(esp)
    addi esp, esp,-4
- Позив процедуре (CALL) – четири микроперације
  - Дохвати PC регистар, смести PC на стек, ажурира показивач, безусловни скок
- Повратак из процедуре (RET) – три микроперације
  - Ажурира показивач, учитај вредност са стека, скочи на срачунату адресу
- Ово је **претпоставка**, може варирати од модела до модела, генерације до генерације

Типови декодера за превођења

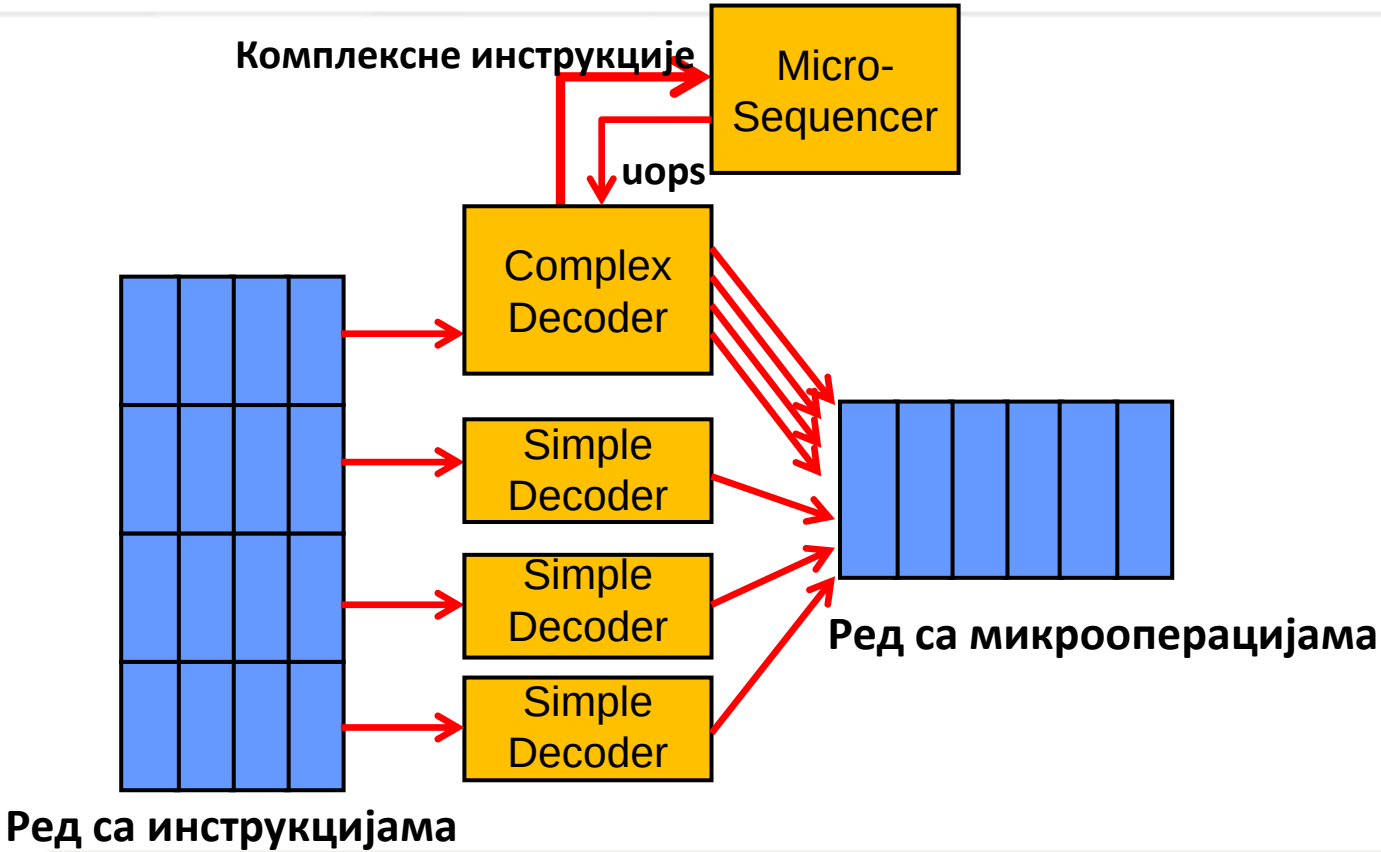
- Уграђена логика: брзи, захтевају додатни хардвер (за инструкције са мало микрооперација)



- Коришћење табела: спори, рачунају са стране и не усложњавају остатак рачунара
  - Обрађују веома сложене инструкције коришћењем микро ROM
  - Користи се за сложене инструкције (> 4 микрооперације), на пример рад са стринговима, или прекидима



Повезивање декодери



Декодери – пример P6

- Интелови декодери
  - Има 3 паралелне декодерске јединице
    - 1 комплексну
    - 2 једноставне
  - Опслужује комплексне / једноставне / једноставне

- АМД (Атлон) декодери
  - Има 3 паралелна декодера
    - Опслужује сваку комбинацију инструкција користећи било који декодер јер су сви декодери истоветни
  - Опслужује и комплексне и једноставне инструкције

Могући начини превођења

- x86 програми почињу да личе на RISC програме
  - Приликом преласка са 32 битне на 64 битну архитектуру удвостручен је број регистара
    - Може се мењати конвенција позивања
    - Више регистара за операнде мање pushe/pop операција
  - Резултат?
    - Мање сложених инструкција
    - Мање микрооперација по x86 инструкцији

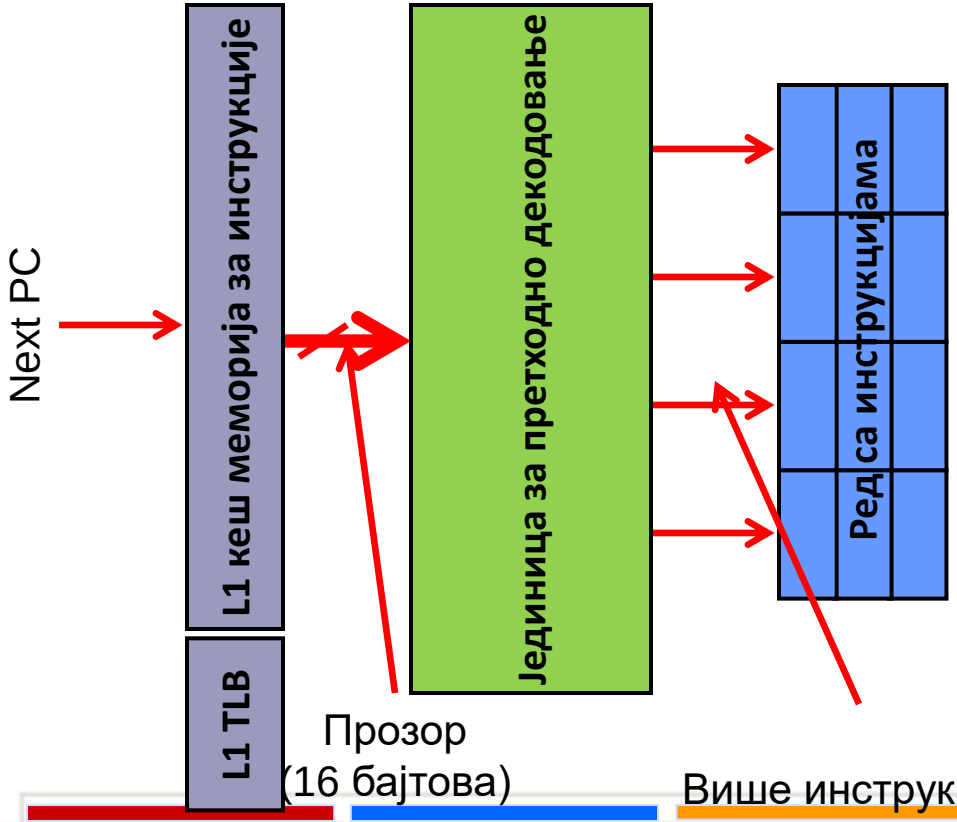
Претходно декодовање инструкција

- Процесори теже да декодирају више инструкције по такту.
- Ако су све инструкције исте дужине, тада их процесор може паралелно обрађивати.
- Ово је принцип *RISC* процесора.
- x86 (*CISC*) инструкције:
  - су комплексне
  - су променљиве дужине (од 1 до 15 бајтова)
  - имају различит број операнада
  - немају конзистентно кодирање
  - одређивање дужине захтева проверу неколико бајтова инструкције
- Тешко је брзо одредити дужину инструкције.
- Потребно је знати дужину једне инструкције како би се знало одакле почиње друга инструкција.
- Због тога је тешко паралелно одредити дужине инструкција.

Претходно декодовање инструкција

- У фази за претходно декодирање (PD) се откривају и обележавају границе између инструкција.
- Из кеш меморије се учитава већи број бајтова података, прозор података (тренутна дужине 16 бајтова).
- У ред са инструкцијама се може уписати више инструкција истовремено (тренутно максимално 6).
- Уписује се од максималног броја инструкција по циклусу или док се не потроше бајтова из прозора, шта год се прво догоди.
- Неће се учитати нови прозор док се претходни прозор потпуно не обради.

Претходно декодовање инструкција



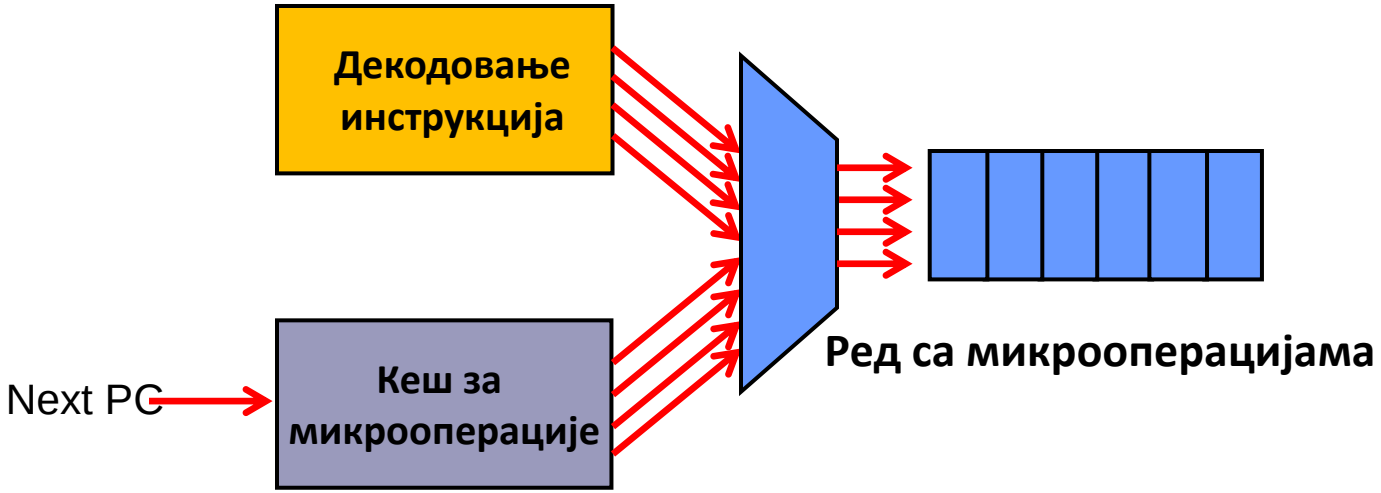
Ред са инструкцијама

- Бафер између јединице за претходно декодирање инструкција и декодера
  - Могућ упис више предекодованих инструкција истовремено
  - Чува више инструкција
  - Могуће читање више инструкција истовремено
  - Свакој инструкцији је дат простор максималне дужине

Кеш за микрооперације

- Инструкције се након што се декодирају у микрооперације чувају у кеш меморији за чување микрооперација.
- Уместо да приступа инструкцијама у инструкцијској кеш меморији нивоа 1, приступа се декодираним микрооперацијама.
- Један важан разлог за то је што је фаза декодирања била уско грло на ранијим процесорима.
- Кеширање микрооперација уместо инструкција омогућава да процесори користе RISC технологију на CISC скупу инструкција.

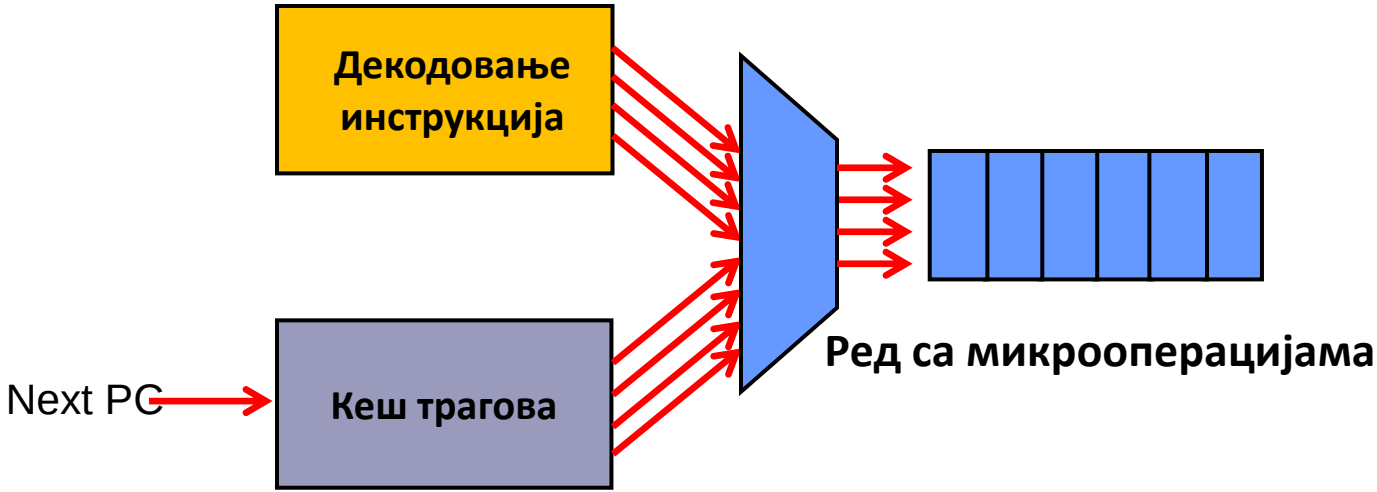
Кеш за микрооперације



Кеш трагова извршавања

- Инструкције заједно са редоследом извршавања се након што се декодирају у микрооперације и изврше могу чувати у кеш меморији за чување трагова.
- Траг у кешу је низ микрооперација које се извршавају у низу, чак и ако нису секвенцијалне у оригиналном коду.
- Предност овога је што је број циклуса такта потрошених на скакање по кешу минималан.
- Веће искоришћење кеш линије.

Кеш трагова извршавања



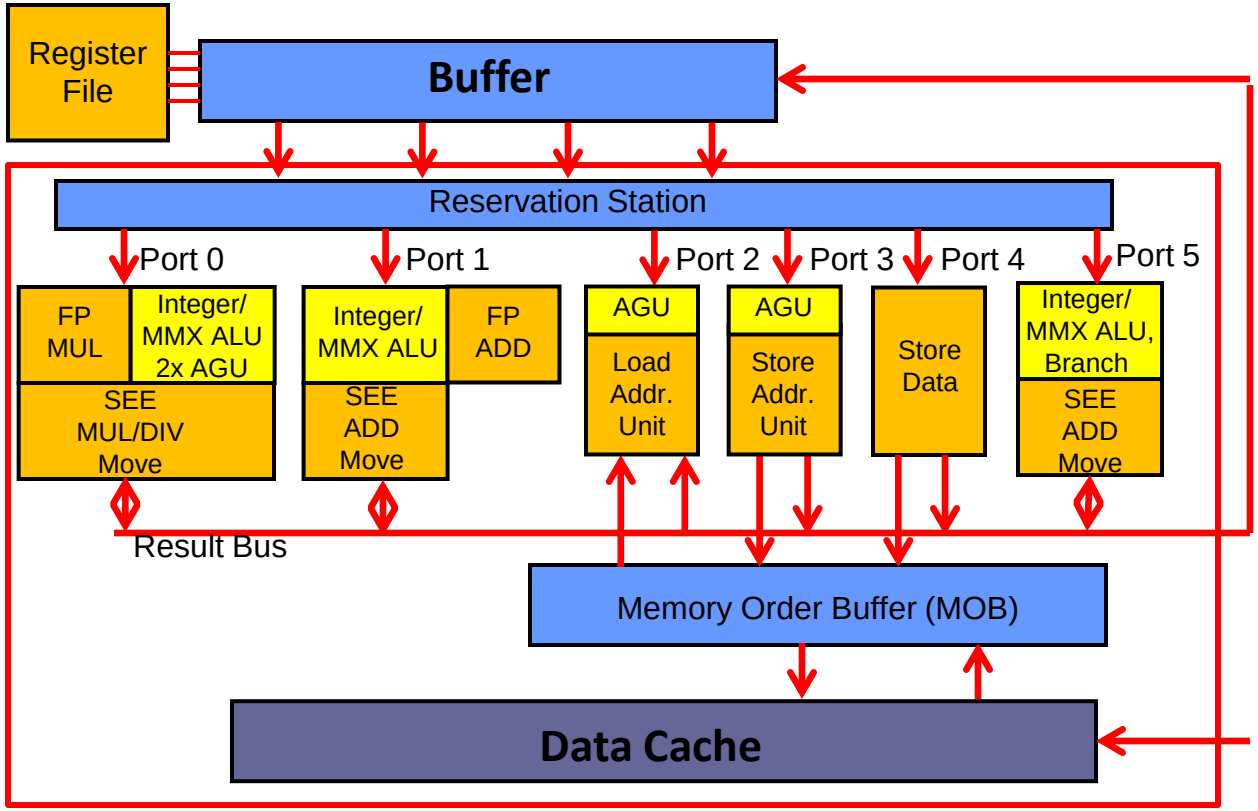
Кеш микрооперација <> Кеш трагова

- Кеш меморија трагова извршавања покушава да замени све претходне степене који користе споро дохватање и декодирање инструкција,
- Кеш меморија за микрооперације се може посматрати као проширење инструкцијеске кеш меморије.

Кеш микрооперација <> Кеш трагова

- Кеш трагова:
  - је скуп и компликован
  - има различите споредне ефекте као што је потреба за испирањем приликом промене контекста.
  - резултира великим дуплирањем
- Кеш микрооперација нема ових недостатака.

Пример јединице за извршавање инструкција



Макрооперације

- Макрооперација је дефинисана као атомска јединица за распоређивање која садржи више инструкција са секвенцијалним редоследом извршења. Извршавање макрооперације мора бити атомично (или у потпуности или никако). То значи да макрооперација може:
  - бити извршена тек након што се задовоље или загарантују да ће бити задовољене све зависности по подацима, тако да се свака инструкција у макрооперацији може довршити без кршења ових зависности дефинисаних у програму, и
  - издавање макрооперације гарантује несметано и детерминистичко извршавање инструкција садржаних у њој.

Макрооперације

- Декодери претварају x86 инструкције променљиве дужине у макрооперације фиксне дужине и достављају их за извршавање према њиховом редоследу доласка *in-order*.
- Ове макрооперације се шаљу разпоређивачима како би се извршиле ван редоследа *out-of-order*.
- Распоређивач претвара макрооперације у микрооперације, које се прослеђују извршним јединицама.
- Извршне јединице могу извршити више микрооперација по такту.
- Задатак декодера је да одржавању тока инструкција, тако да се никад не заустави због недостатка макрооперација које се шаљу на извршавање.

Макрооперације – Интел

- **Интел:** x86 инструкције променљиве дужине.
  - У њиховом контексту, макрооперације су променљиве дужине и могу бити прилично сложене и способне да извршавају више меморијских и аритметичких операција одједном.
  - У АМД-овом контексту, оне се називају стварним „АМД64 инструкције”.

Макрооперације – АМД

- **АМД:** 1. поједностављене инструкције фиксне дужине (понекад и сложеније инструкције).
  - У њиховом контексту, макрооперација је инструкција фиксне дужине која се може састојати од операције приступа меморије и аритметичке операције.
  - На пример, један макрооперација може извршити операцију читања, модификовања и писања.
- 2. инструкције која су прошле низ трансформација како би се уклопиле у строжи, али ипак сложени облик.
  - У Интеловом контексту, такав концепт не постоји.

Микрооперације / Макрооперације – АМД

Поређење	АМД64 инструкције	Макрооперације	Микрооперације
Комплексност	Комплексне Једна инструкција може специфицирати једну или више следећих операција: • Integer / floating-point • Load • Store	Просечне Једна макрооперација може специфицирати највише једну integer / floating-point операцију и једну од следећих операција: • Load • Store • Load + Store на исту адресу	Једноставне Једна микрооперација специфицира само једну од следећих основних операција: • Integer / floating-point • Load • Store
Дужин инструкције	Променљива (инструкције су различите дужине)	Фиксна (све макрооперације су исте дужине)	Фиксна (све микрооперације су исте дужине)
Правилност у распореду поља	Не (распоред поља и дефиниције варирају међу инструкцијама)	Да (фиксна локација поља и њихова дефиниција за све макропоперације)	Да (фиксна локација поља и њихова дефиниција за све микропоперације)

Макрооперације – АРМ

- **АРМ:** сложене инструкције које се разлажу на одређени број микрооперација којима је потребан низ итерација кроз проточну обраду са више циклуса.
  - Такође и неке друге сложене инструкције које се разлажу на мање микрооперације се могу сматрати макрооперацијама
- У њиховом контексту, не користи се експлицитно овај израз.

Спајање операција

- Спајање микрооперација
  - спајање микрооперација које се понављају у програму (спајање парова load/add, спајање смештање адресе и податка)
  - Учитавање микрооперације се може смањити за 10%
- Спајање макрооперација
  - спајање самих инструкција, не само микрооперација. (пример: спајање поређења и скока код x86)
  - Учитавање инструкција се може смањити за 15%

Спајање операција

- Неке фазе у проточној обради су уска грла са максималном пропусношћу у броју микрооперације по такту. Да би се постигао већи проток кроз ова уска грла, удружује се неке операције које проистичу из исте инструкције а које су у претходним процесорима биле подељене у две.
- Спајање омогућава да оно што би биле (две) одвојене операције (микрооперације или макрооперације), комбинују се или се стапају у једну микрооперацију.

Спајање операција

- Спојене операције деле једну микрооперацију у већини степени проточне обраде и један унос у бафер за преуређивање редоследа (*reorder buffer - ROB*).
  - Ако овај запис у баферу представља две операције које морају да изврше две различите извршне јединице онда се спојени запис шаље на два различита извршна порта, али мора да се заврши (*retire/graduate*) као једна операција.
  - Ако овај запис у баферу представља две операције које може да изврши једна извршна јединица онда се спојени запис шаље на тај један извршни порта, и завршава се као једна операција.

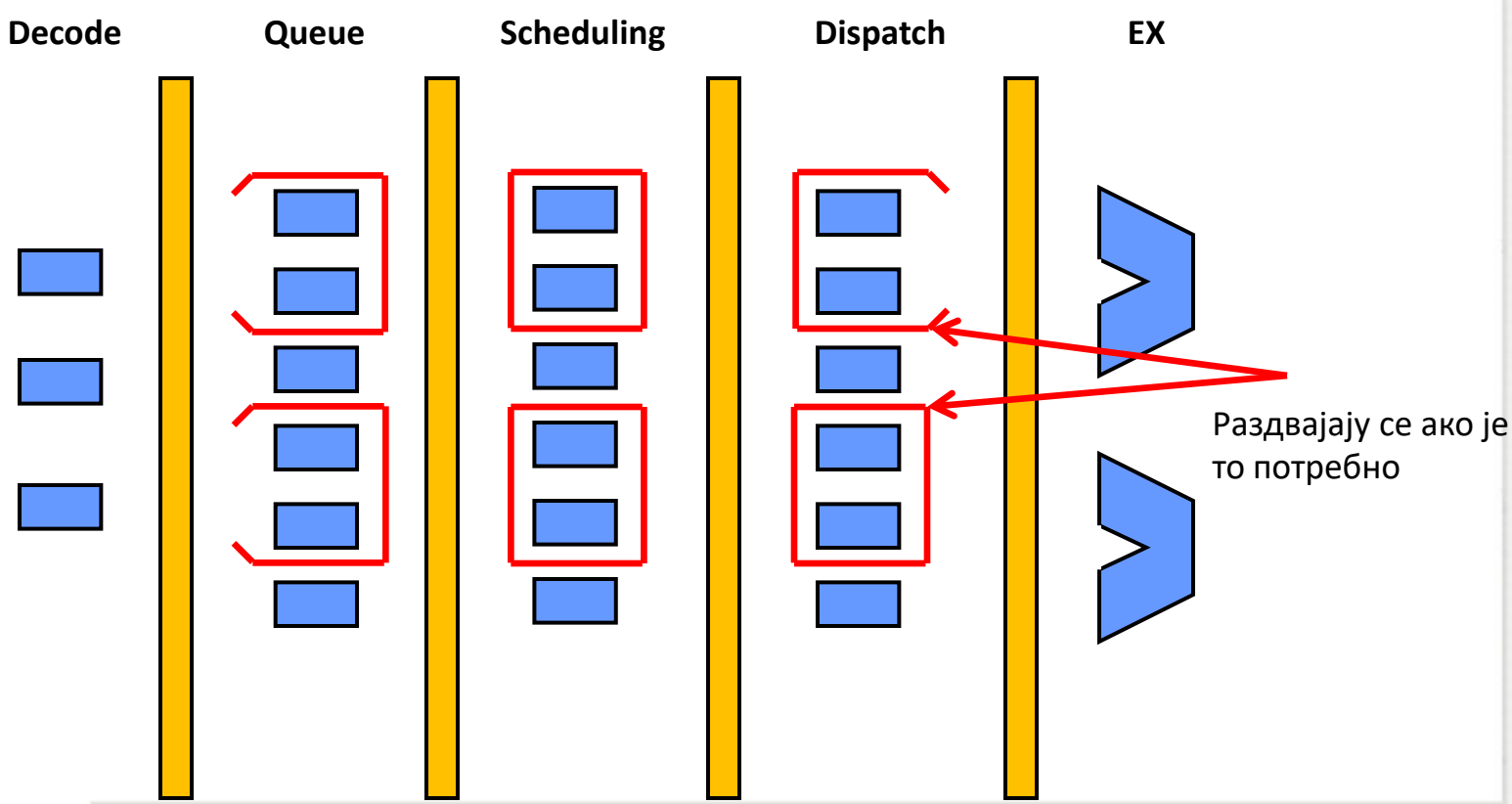
Спајање операција

- Са спајањем операција, декодер једноставно емитује једну микрооперацију која заузима само један запис у свакој од претходно поменутих структура.
- Током распореда и извршења, две половине микрооперације **могу** се издавати и извршавати независно као да су у ствари две одвојене, различите, традиционалне микрооперације.

Спајање микрооперација

- Техника спајања микрооперација се тренутно може применити код две комбинације микрооперација:
  - Операција које уписују у меморију
  - Операција које чита и мења операнд

Спајање микрооперација



Спајање микрооперација

- Операција уписивања у меморију најчешће укључује кораке израчунавање адресе и пренос података.
- На пример: **mov [mem32], eax**
  На неким процесорима су ова два корака подељени у две микрооперације, где једна јединица (порт 3) брине о израчунавању адресе, а друга јединица (порт 4) о преносу података.

- Операција читања меморије најчешће захтева само једну микрооперацију то јест једну јединицу (порт 2).

[mem32] означава неку адресу у меморији, не и начин адресирања

Спајање микрооперација

- Друга врста операција која се може спојити је операција читања-модификовања.
- На пример:

  add eax, [mem32] - 2 микрооперације.
  Прва микрооперација чита се из меморије (порт 2), друга микрооперација (порт 0 или 1) додаје вредност која је прочитана у **eax**. Таква инструкција је подељена у две микрооперације на претходним процесорима, али се могу спојити.
- Ово се односи на мноштво инструкција за читање и модификовање које раде на регистрима опште намене, регистром стека са покретним зарезом и MMX регистрима, али не и на инструкције за читање и модификовање који раде на XMM регистрима.

Спајање микрооперација

- Инструкције читања-модификовања-писања, као што је **add [mem32], eax** не спаја читање и модификовање микрооперација, али спаја две микрооперације потребне за писање.

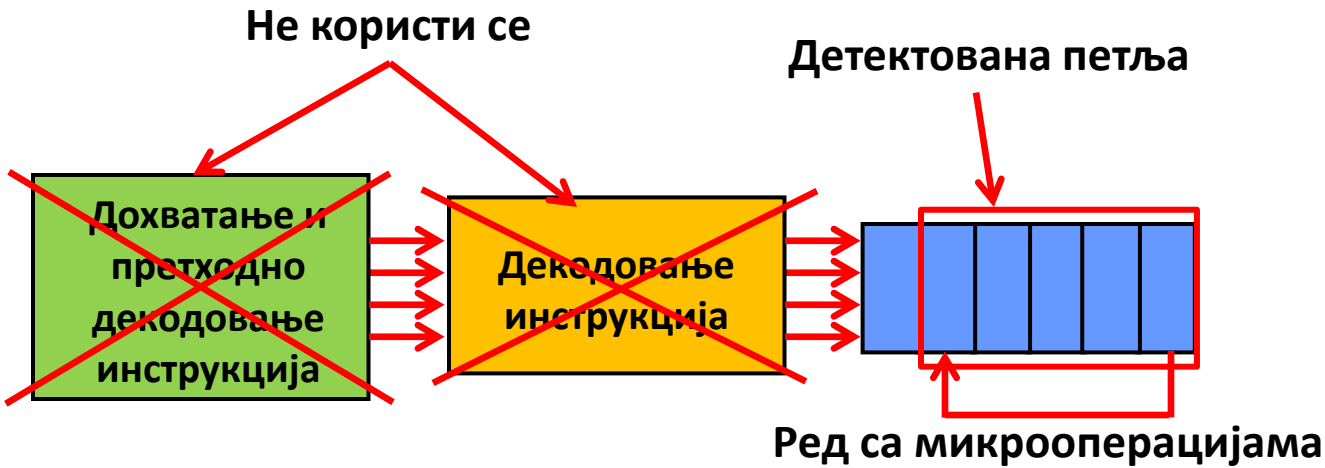
Спајање микрооперација

- Пример спајања микрооперација:
  - **mov (esi), eax**; 1 спојена uop
  - **add eax, (esi)**; 1 спојена uop
  - **add (esi), eax**; 2 појединачне + 1 спојена uop
  - **fadd qword ptr (esi)**; 1 спојена uop
  - **paddw mm0, qword ptr (esi)**; 1 спојена uop
  - **paddw xmm0, xmmword ptr (esi)**; 4 uop, нема спајања
  - **addss xmm0, dword ptr (esi)**; 2 uop, нема спајања
  - **movaps xmmword ptr (esi), xmm0**; 2 спојене uop

Емитовање детектованих петљи

- Детектовање петљи (*Loop Stream Detector - LSD*)
  - Овај механизам може директно да емитује микрооперације без потребе за поновним дохватањем или декодовањем.
  - Ово је једноставан начин за уштеду енергије јер док је механизам активан остатак предњег дела је неактиван (и декодери и кеш микрооперација).
- Да би се искористио овај механизам петља треба да буде мала тако да стане у прописани број микрооперација.

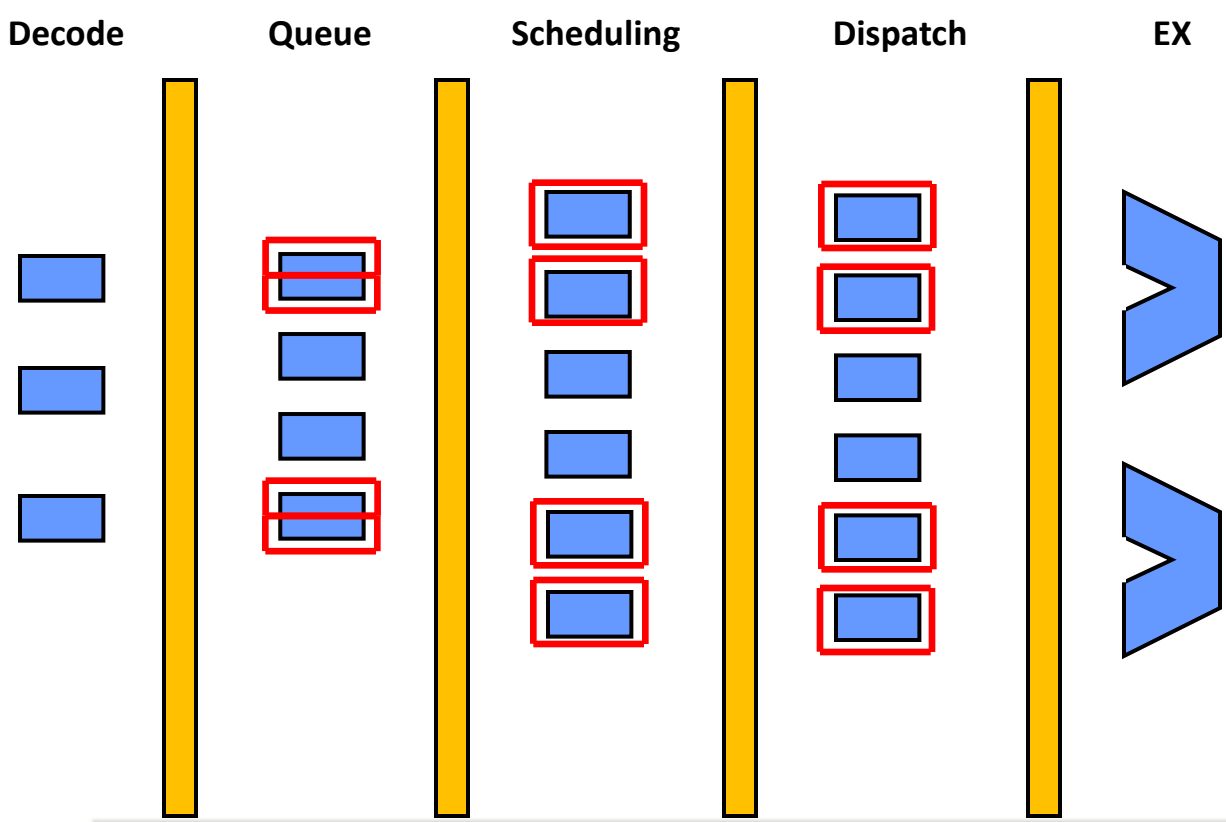
Емитовање детектованих петљи



Раздвајање микрооперација

- Ред за микрооперације пружа функцију накнадног декодирања за одређене типове инструкција. Конкретно, учитавање у комбинацији са рачунским операцијама и сва смештања, када се користи са индексираним адресирањем, представљена су као једна микрооперација у декодеру или кеш меморији.
- У реду за микрооперације они су раздвојени у два микрооперације кроз процес који се назива раздвајање (*un-lamination*), један врши учитавање, а други операцију.

Раздвајање микрооперација



Раздвајање микрооперација

- Типичан пример је:

  add ax, (bp+si); ax := ax + ld(bp+si)

- Слично је и са инструкцијом која користи три регистра која се раздваја на инструкције за генерисање адресе и генерисање податка:

  mov (sp+cx*4+12345678), al

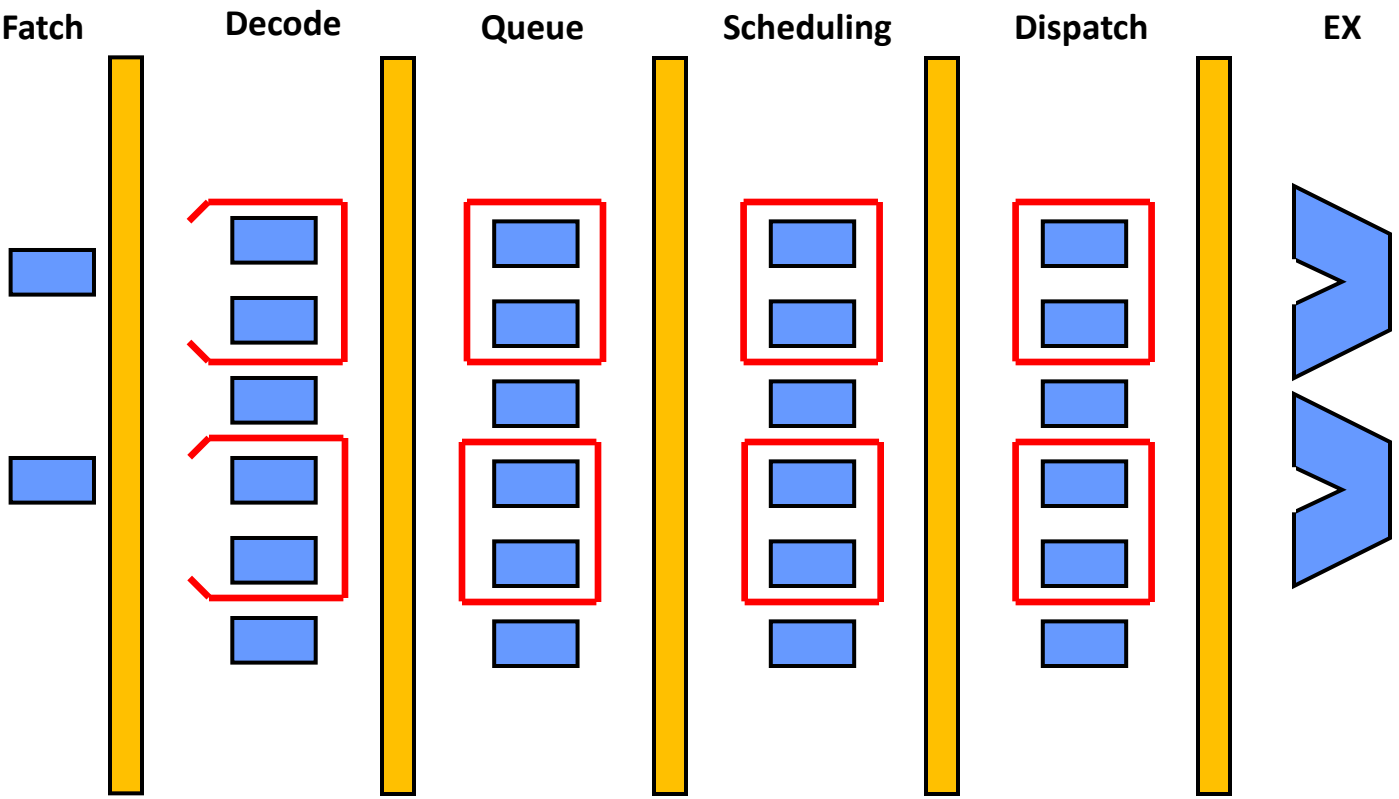
Раздвајање микрооперација

- Додатне микрооперације генерисане раздвајањем користе више ресурса у фази преименовања и завршетка инструкције. Међутим, то има укупну корист по потрошњу струје и о број искоришћених транзистора (заједно са главном променом у коришћењу физичког регистрског фајла, уместо да се улазни/излазни подаци чувају у беферу за промену редоследа).
- За код којим доминира индексирано адресирање (као што се често дешава код обраде низа), алгоритми за кодирање да би се користило базно (или базно са померајем) адресирање могу понекад побољшати перформансе одржавањем дохватања и обраде и инструкције за чување стопљеним.

Спајање макрооперација

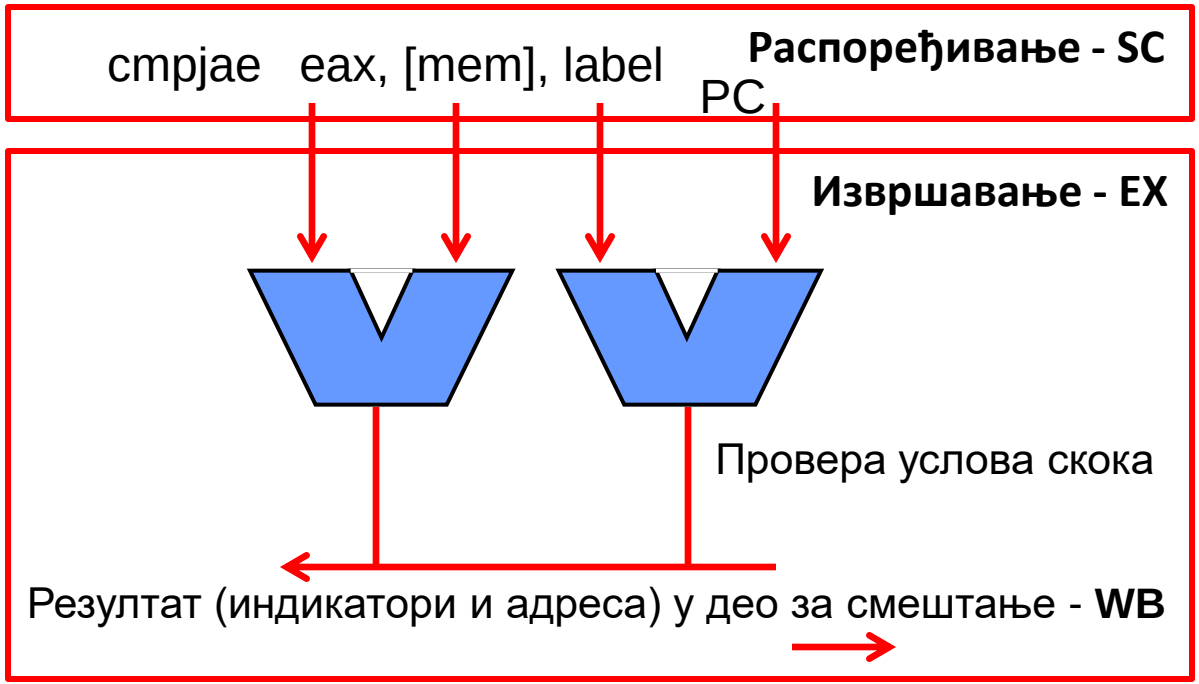
- Уобичајени парови инструкција комбиновани у јединствену макрооперацију:
  - упоређивање праћено условним скоком одвија се током фазе декодирања
- Коришћење побољшане АЛУ (ЕАЛУ)
  - једноциклично извршавање комбинованих парова инструкција
- Проширено на спајање микрооперација
  - макрооперације подељене на мање микрооперације
  - микрооперације спојене и извршене заједно
  - смањује број микрооперација којима се рукује ван редоследа за више од 10 посто

Спајање макрооперација



Спајање макрооперација

- Побољшане АЛУ (ЕАЛУ)



Спајање макрооперација – x86

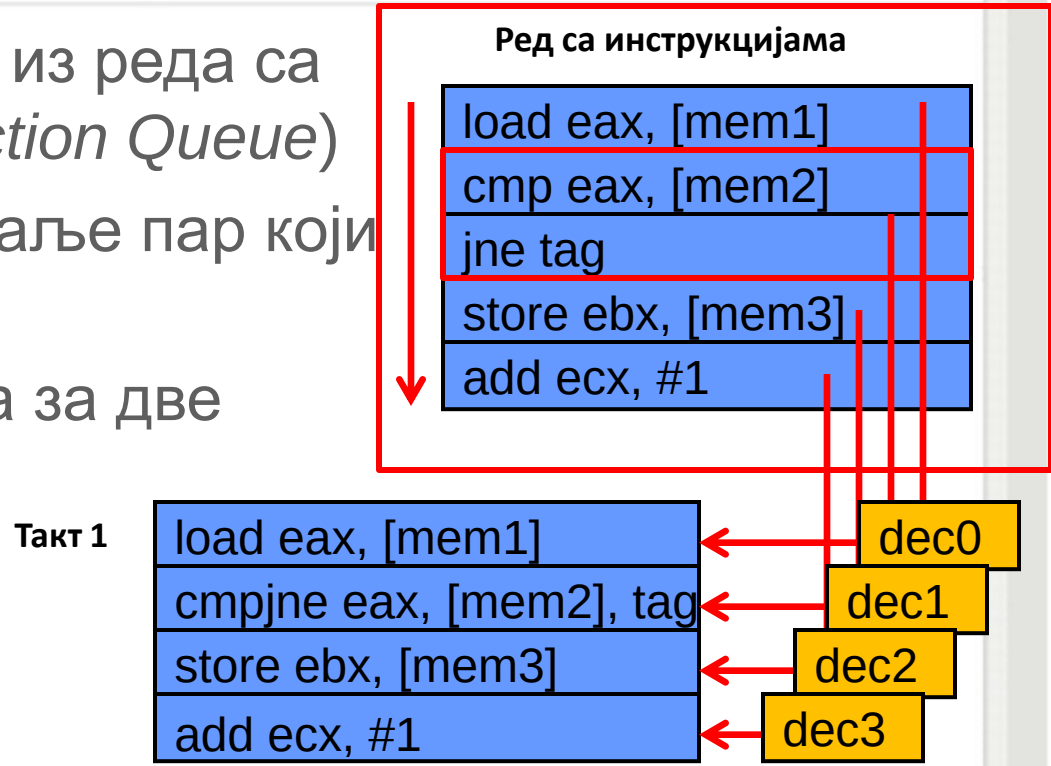
- Декодери ће спојити инструкције поређења (*CMP* и *TST*) са наредном условним скоком у једну микрооперацију упоређивања и гранања у одређеним случајевима. Микрооперација упоређивање и гранање се не дели на два дела на извршним јединицама, већ се на извршном порту извршава као јединствену микрооперацију. То значи да фузија макрооперације штеди пропусни опсег у свим фазама проточне обраде од декодирања до завршетка.
- Спајање макрооперација, међутим, не помаже ако је фаза претходног декодовања (*PD*) уско грло.

Спајање макрооперација – x86

- Спајање макрооперација је (тренутно) могућа само ако су испуњени сви следећи услови:
  - Прва инструкција је *CMP* или *TST* инструкција, а друга инструкција је инструкција условног скока, осим *JECXZ* и *LOOP*.
  - *CMP* или *TST* инструкција може да има два операнда регистра, регистар и непосредни операнд, регистар и меморијски операнд, али не и меморију и непосредни операнд.

Спајање макрооперација – има спајања

- Чита се **5** инструкције из реда са инструкцијама (*Instruction Queue*)
- На исти декодер се шаље пар који се може спојити
- Једна микрооперација за две инструкције



Спајање макрооперација – x86

- Гране које проверавају бите Z и C (*JE, JNE, JB, JBE, JA, JAE*) могу се стопити са претходним *CMP* или *TST* . То укључује сва **неозначена** поређења. Гране за означеним поређења (*JL, JLE, JG, JGE*) могу се спојити са претходним *CMP* или *TST* на неким језгрима (*Core Nehalem*), али само са *TST* на језгру *Core 2*.
- Гране које проверавају бите за преливање, паритет или знак (*JO, JNO, JP, JNP, JS, JNS*) могу се спојити са *TST*, али не и са *CMP*.
- Између две инструкције се не смеју наћи друге инструкције.
- Условни скокови не би требало да започиње на граници прозора који се у једном такту учитава из кеш меморије (16 бајтова) нити да прелази ту границу.
- Ако више од једног таквог пара инструкција достигне истовремено фазу декодовања само се први спаја.

Спојивост макрооперација – x86

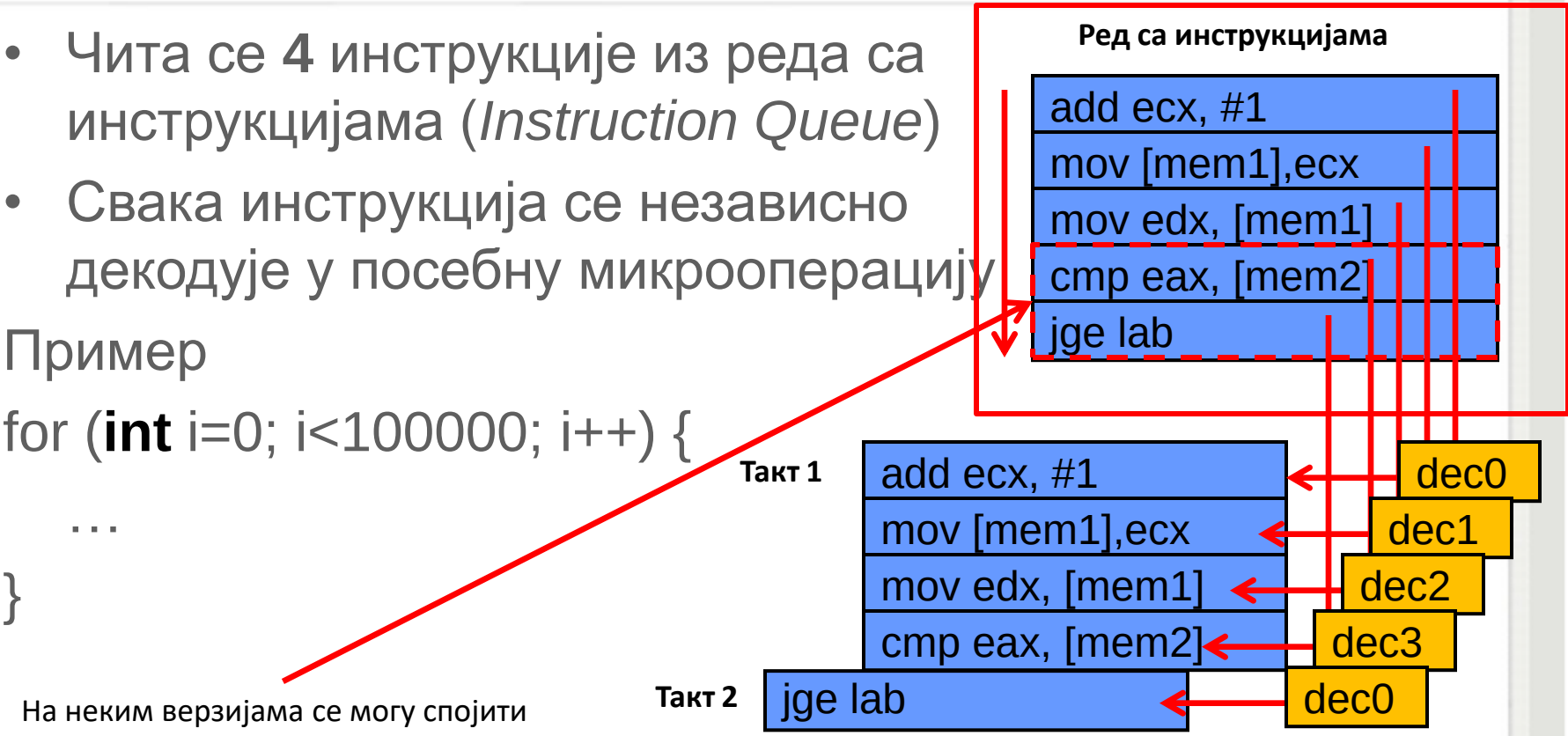
Instruction	TEST	CMP	AND	ADD	SUB	INC	DEC
JO/JNO	+	-	+	-	-	-	-
JC/JB/JAE/JNB	+	+	+	+	+	-	-
JE/JZ/JNE/JNZ	+	+	+	+	+	+	+
JNA/JBE/JA/JNBE	+	+	+	+	+	-	-
JS/JNS/JP/JPE/JNP/JPO	+	-	+	-	-	-	-
JL/JNGE/JGE/JNL/JLE/JNG/JG/JNLE	+	+	+	+	+	+	+

Спајање макрооперација – нема спајања

- Чита се **4** инструкције из реда са инструкцијама (*Instruction Queue*)
- Свака инструкција се независно декодује у посебну микрооперацију

Пример

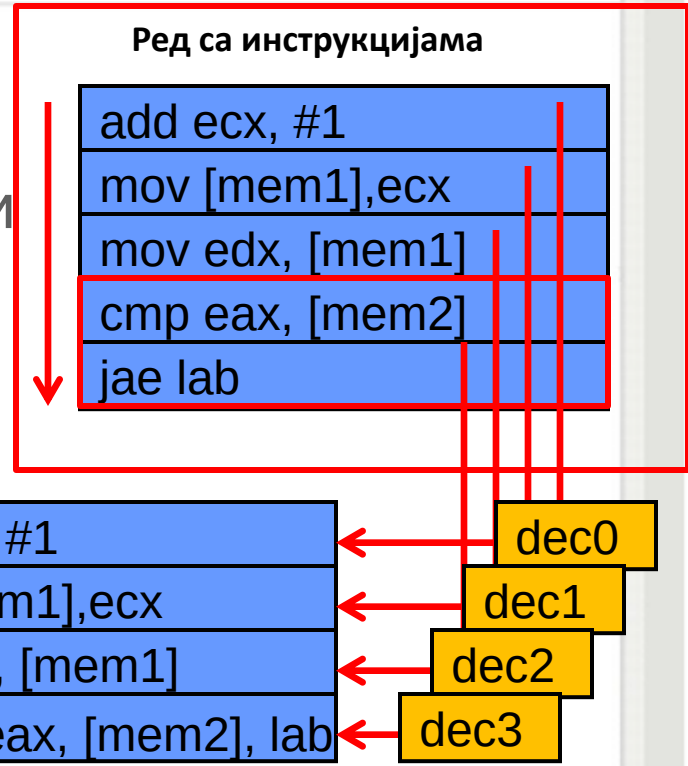
```
for (int i=0; i<100000; i++) {
    …
}
```



Спајање макрооперација – има спајања

- Чита се **5** инструкције из реда са инструкцијама (*Instruction Queue*)
- На исти декодер се шаље пар који се може спојити
- Једна микрооперација за две инструкције
- Пример

```
for (unsigned int i=0; i<100000
    …
}
```



Intel Core i7

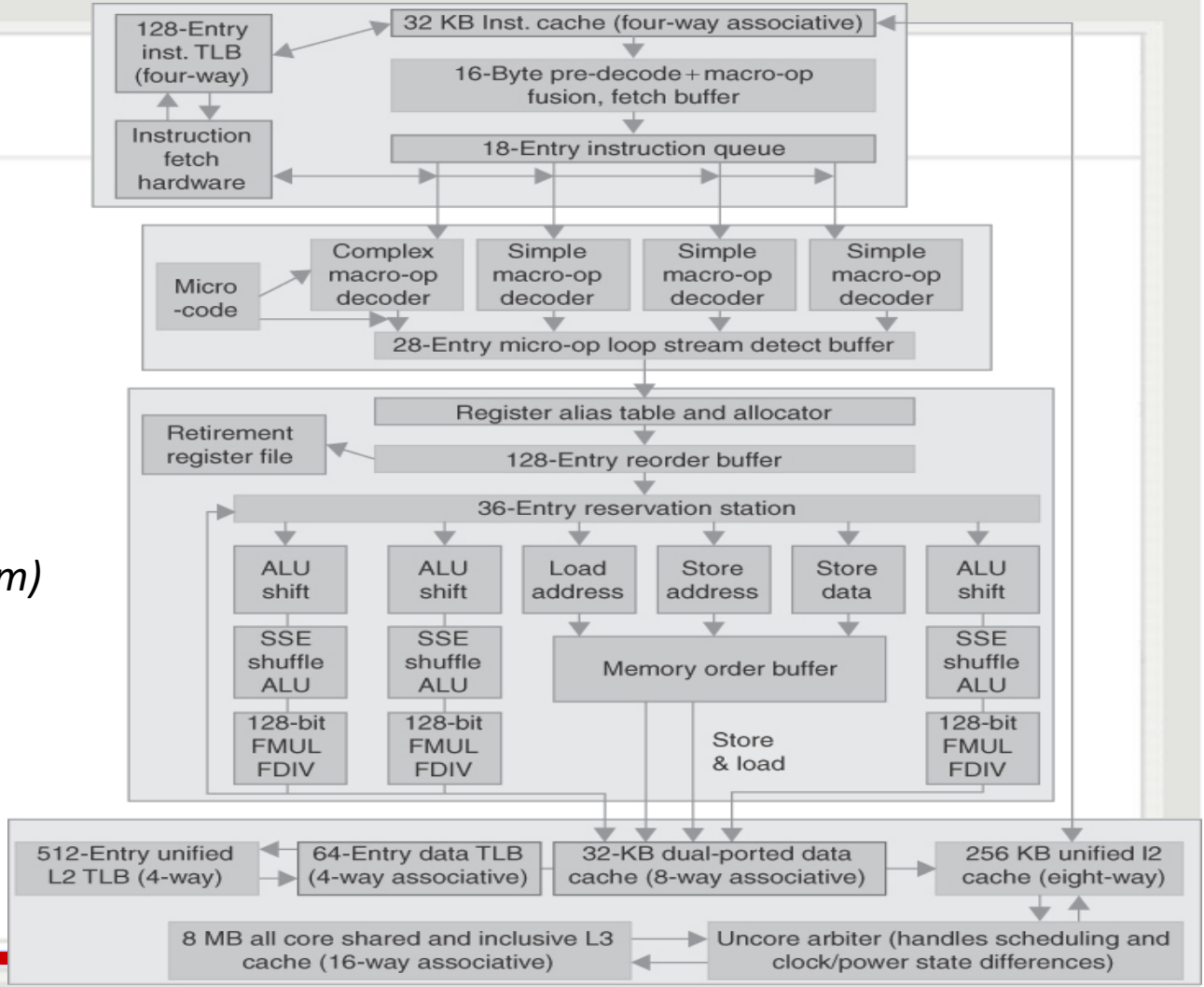
- i7 проширује Пентиумов приступу
- Користи агресивну спекулацију за извршавање ван редоследа
- Користи дубоку проточну обраду (14 фаза)
  - Дохватање инструкција – дохвата 16 бајтова за декодирање
  - постоји засебна јединица (*Integrated Instruction Fetch Unit – IIFU*) који пуни ред који може истовремено да ускладишти до 18 инструкција
  - за разлику од Пентиума, декодирање се врши помоћу корака који се назива фузија макрооперација који комбинује инструкције које имају независне микрооперације које се могу извршавати паралелно
  - ако се открије петља која садржи мање од 28 инструкција или 256 бајтова, ове инструкције ће остати у баферу за вишекратно издавање (уместо поновљених дохвата инструкција)

Intel Core i7

- Дохватање инструкција такође укључује
  - Коришћење бафера скокова са више нивоа и стека повратне адресе
    - Промашено предвиђање скока носи казну од око15 циклуса
  - Кеш меморија инструкција од 32 KB
- Декодирање прво претвара машинске инструкције у микрооперације и разбија их у два типа помоћу четири декодера
  - Једноставна микрооперације (по 3)
  - Сложене микрооперације (по 1)
- Издавање инструкције може издати до 6 микрооперација по циклусу
  - 36-97 централизованих резервационих станица
  - 6 функционалних јединица, укључујући 1 јединицу за учитавање и 2 јединице за складиштење које деле бафер меморије повезан на 3 различите кеш меморије података

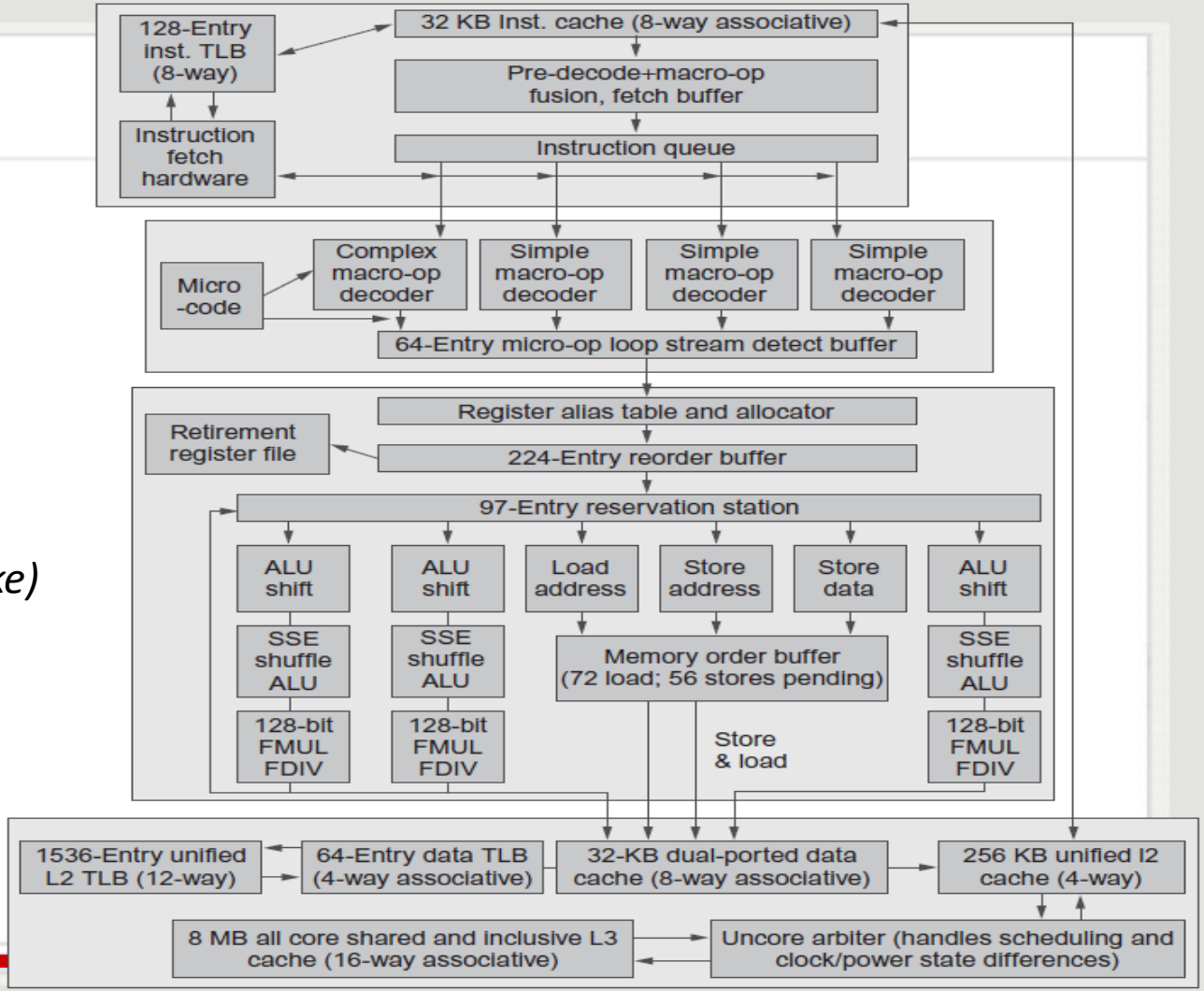
Intel Core i7

i7- 920 (Nehalem)



Intel Core i7

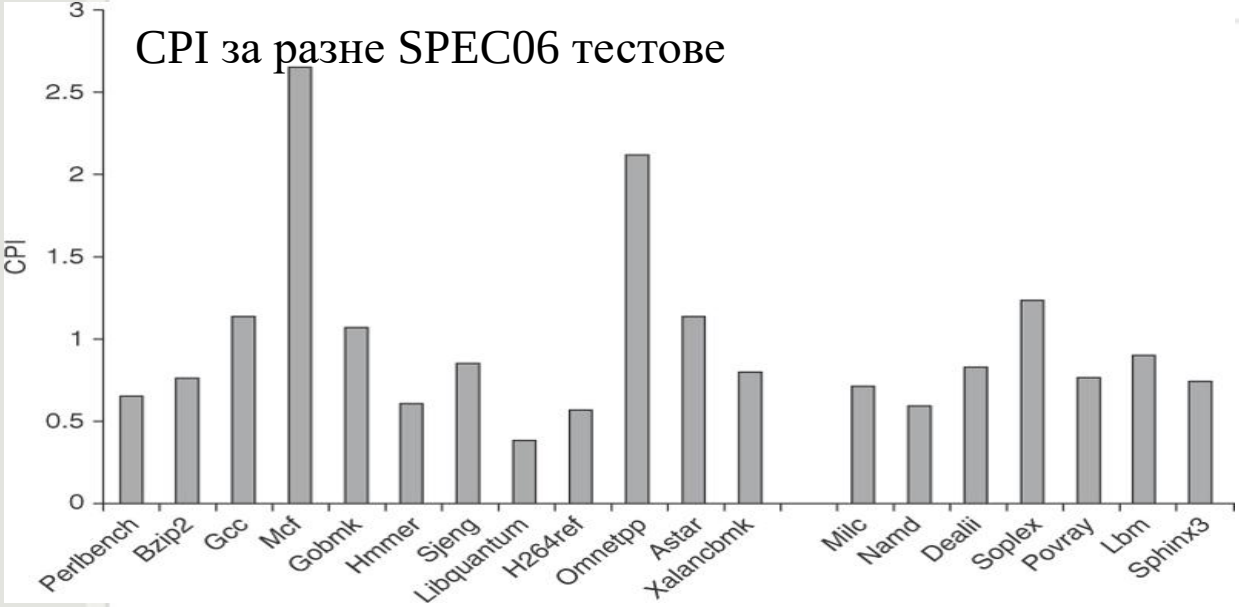
i7- 6700 (Skylake)



i7 920 (Nehalem) vs i7 6700 (Skylake)

Resource	i7 920 (Nehalem)	i7 6700 (Skylake)
Micro-op queue (per thread)	28	64
Reservation stations	36	97
Integer registers	NA	180
FP registers	NA	168
Outstanding load buffer	48	72
Outstanding store buffer	32	56
Reorder buffer	128	224

Intel Core i7 - перформансе

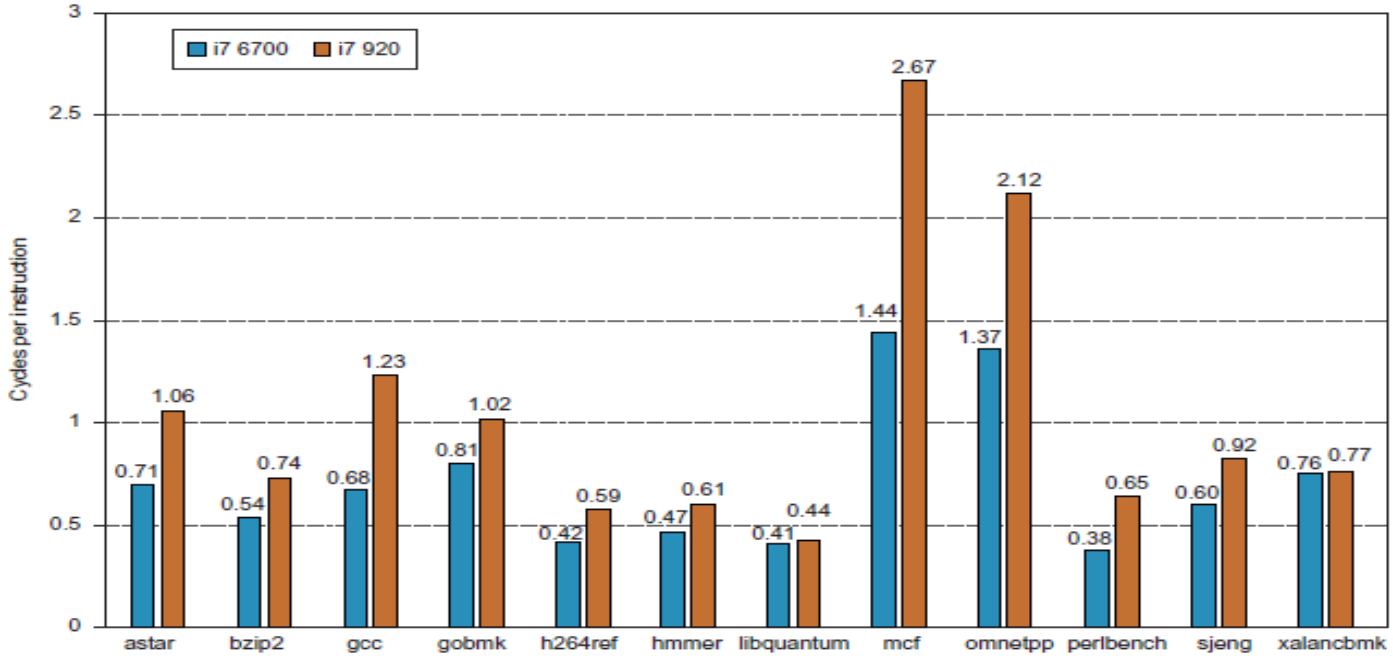


Просечни CPI је 1,06 за целобројне програме и 0,89 за оне у покретном зарезу. Ово је број покренутих машинских инструкција (не микрооперације) па добијање вредности није потпуно разлучив.

Пентиум и i7 су подложни нагађањима, што резултира изгубљеним радом, изгуби се до 40% укупног рада који улази у Specs06 тест.

Губитак настаје и због промашаја у кешу (10 циклуса или више изгубљених са промашајем у L1, и 30-40 за L2 промашаје и чак до 135 за L3 промашаје)

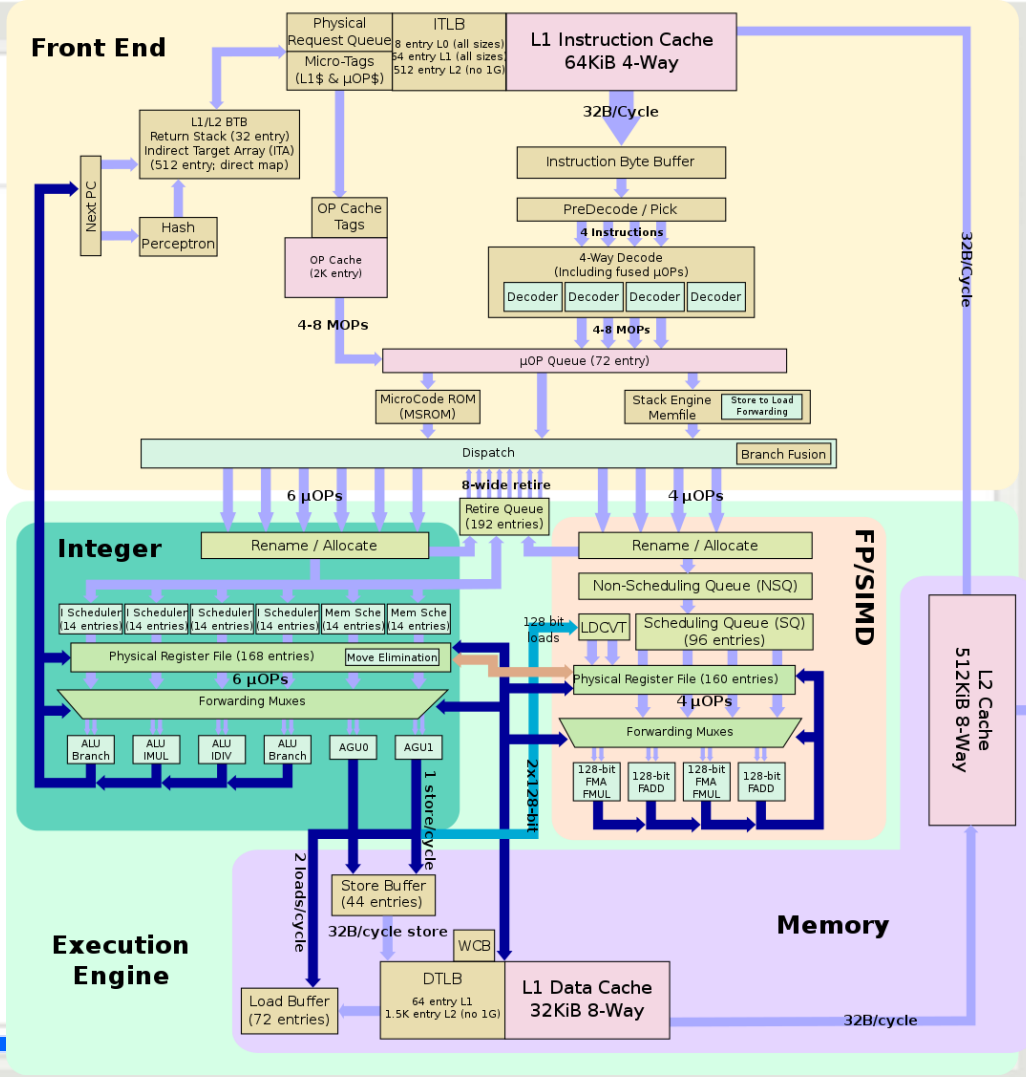
i7 920 (Nehalem) vs i7 6700 (Skylake)



i7 920 (Nehalem) vs i7 6700 (Skylake)

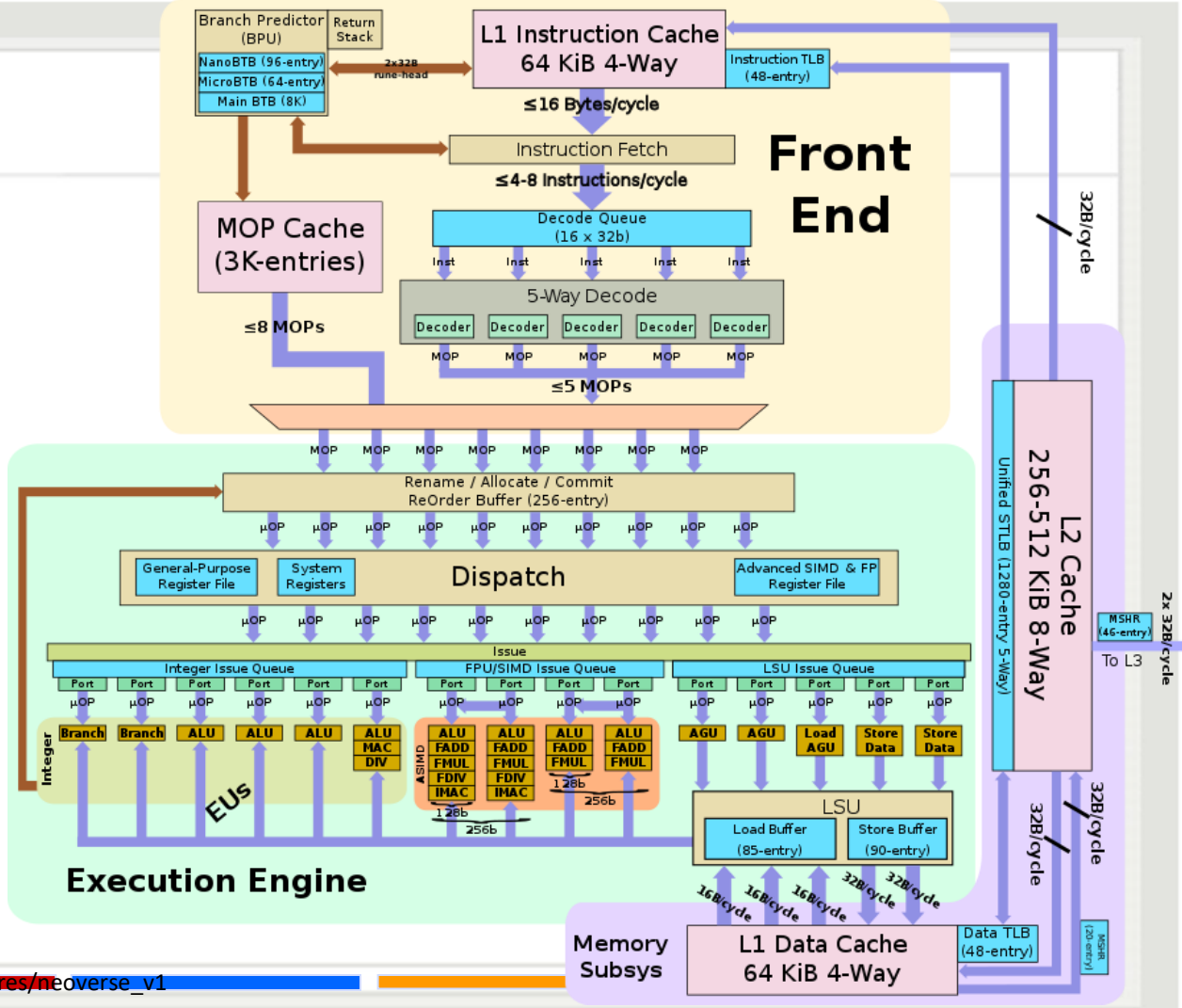
Тест	Однос CPI (920/6700)	Однос лошег предвиђања скокова (920/6700)	Однос проомашаја код L1 кеш меморије (920/6700)
ASTAR	1,51	1,53	2,14
GCC	1,82	2,54	1,82
MCF	1,85	1.27	1,71
OMNETPP	1,55	1,48	1,96
PERLBENCH	1,7	2,11	1,78

AMD - Zen



https://en.wikichip.org/wiki/amd/microarchitectures/zen

ARM - Zeus



https://en.wikichip.org/wiki/arm_holdings/microarchitectures/neoverse_v1

Питања?

Електротехнички Факултет
Универзитет у Београду