Архитектура рачунара

Садржај

- Типови података
- Формати инструкција
- Скуп инструкција
- Програмски доступни регистри
- Начини адресирања

Структура рачунара

- Операције које се извршавају у рачунару се представљају помоћу бинарних речи које се називају инструкције, команде или наредбе
- Било који проблем који треба да се решава у рачунару може да се разложи на уређени низ инструкција који се назива програм
- Подаци над којима се операције извршавају се такође представљају помоћу бинарних речи које се називају операнди

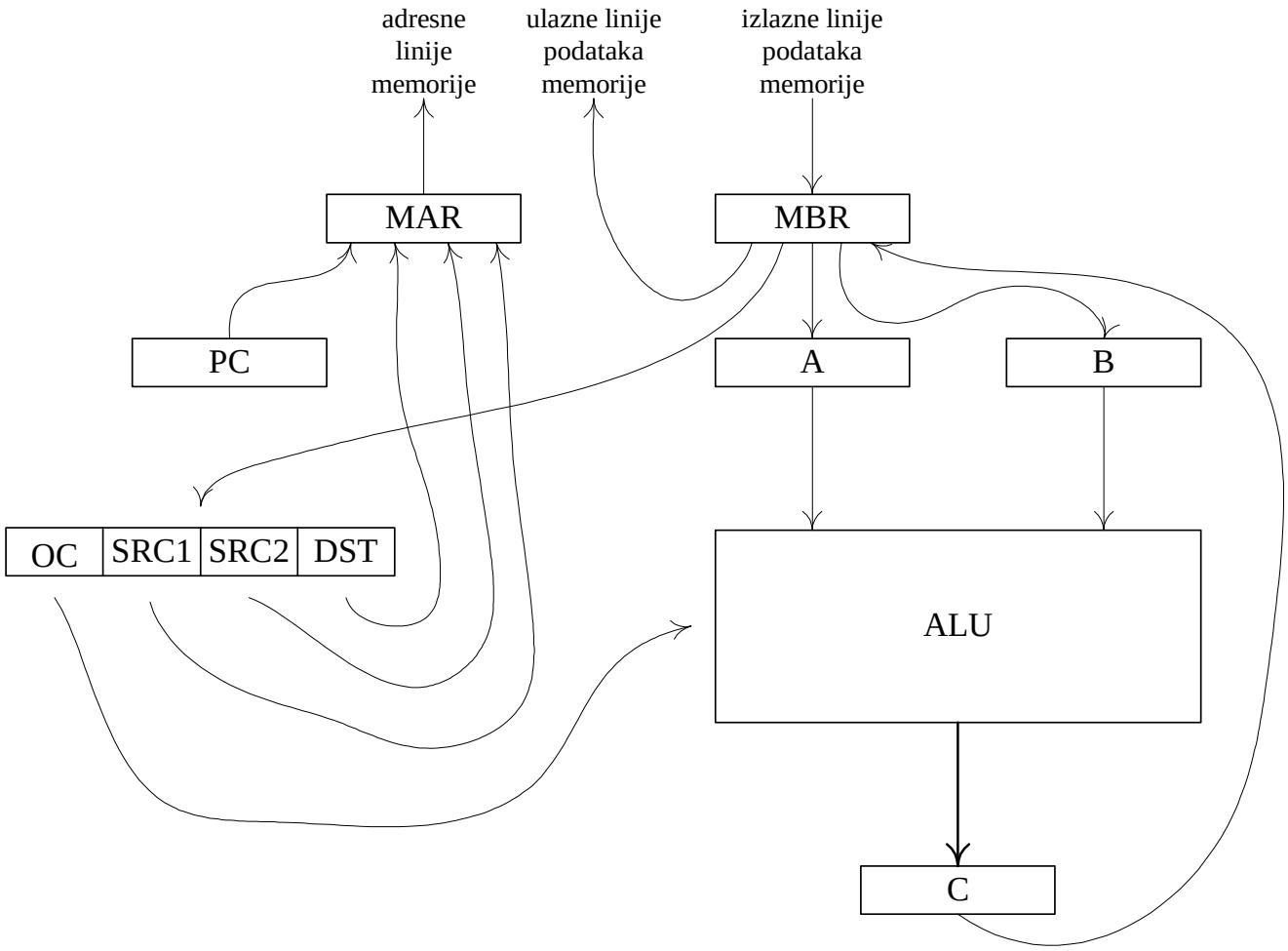
Меморија

- За складиштење оба типа речи користи се модул рачунара који се зове меморија

- Y=(A-B)/[C+(D·E)]

SUB	A,	B,	Y
MUL	D,	E,	T
ADD	T,	C,	T
DIV	Y,	T,	Y

Процесор


adresne
linije
memorije
ulazne linije
podataka
memorije
izlazne linije
podataka
memorije
MAR
MBR
PC
A
B
OC
SRC1
SRC2
DST
ALU
C

Дијаграм тока извршавања инструкције

Типови података

- Најчешће коришћени типови података су:
  - целобројне величине
  - величине у покретном зарезу
  - алфанумерички низ
  - нумерички низ.

Интерпретација бинарне речи

- Ако се бинарна реч дужине n битова, у којој су битови означени са $a_{n-1}a_{n-2}...a_1a_0$, интерпретира као целобројна величина без знака, онда она представља податак A чија се вредност израчунава помоћу израза

$$A = \sum_{i=0}^{n-1} 2^i a_i$$

- Уз такав начин интерпретирања битова бинарне речи, представљају се целобројне величине без знака у опсегу 0 до 2^n-1.

Интерпретација бинарне речи

- Међутим, ако се иста бинарна реч интерпретира као целобројна величина са знаком у другом комплементу, онда она представља податак A чија се вредност израчунава помоћу израза

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

- Уз овакав начин интерпретирања битова бинарне речи, представљају се целобројне вредности са знаком у опсегу -2^{n-1} до $2^{n-1}-1$.

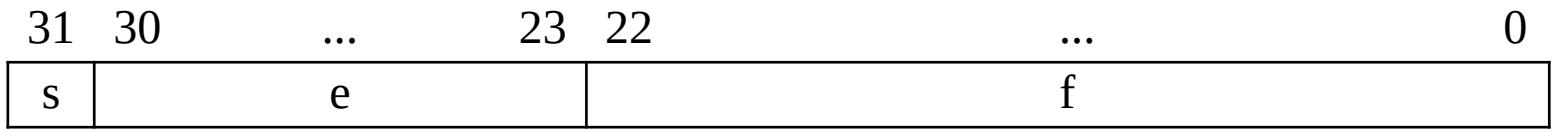
Интерпретација бинарне речи

- Целобројне величине фиксне дужине се у рачунарима представљају на фиксним дужинама од 8, 16, 32 и 64 бита и њихова дужина и начин интерпретације су одређени пољем кода операције инструкције.

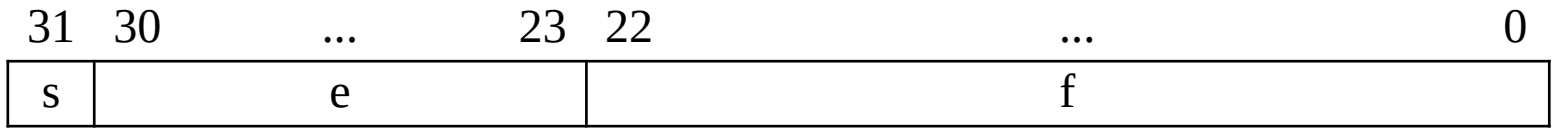
Величине у покретном зарезу

- Величине у покретном зарезу имају:
  - поље знака (s),
  - поље експонента (e),
  - поље мантисе (f).
- Дужине су 32 или 64 бита.

- Пример:

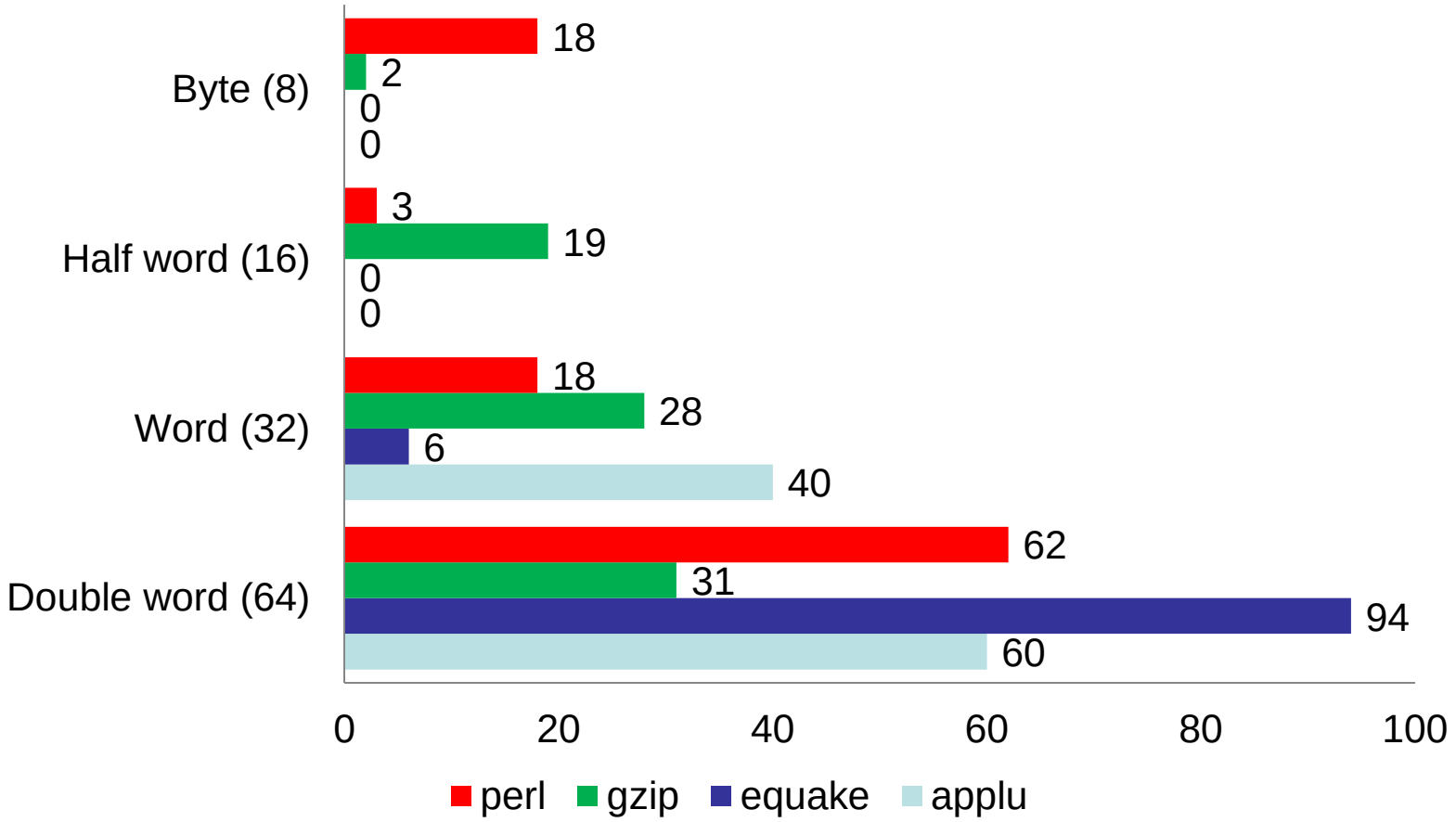


Величине у покретном зарезу



- Вредност се добија на следећи начин:
  - за e=255 и f≠0, v је Not a Number без обзира на s,
  - за e=255 и f=0, $v=(-1)^s \infty$,
  - за 0<e<255, $v=(-1)^s 2^{e-127} (1.f)$,
  - за e=0 и f≠0, $v=(-1)^s 2^{e-126} (0.f)$ – денормализовани број,
  - за e=0 и f=0, $v=(-1)^s 0$ (нула).

Величина податка (операнда)



Формати инструкција

- Форматом инструкције се специфицирају две врсте информација неопходне за извршавање инструкција програма и то:
  - операција и тип податка са којим операција треба да се реализује и
  - изворишни и одредишни операнди.

Изворишни и одредишни операнди

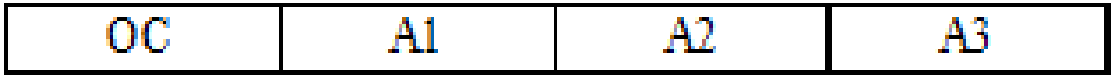
- Овим пољем се експлицитно специфицирају операнди. Постоје више варијанти овог поља које настају као последица следећа два елемента:
  - број експлицитно специфицираних операнада и
  - могуће локације операнада.

Број експлицитно специфицираних операнада

- Адресни формати инструкција
  - Троадресни – три експлицитна операнда
  - Двоадресни – два експлицитна операнда
  - Једноадресни – један експлицитан операнд
  - Нулаадресни – нема експлицитних операнада (?)

Троадресни формат инструкције

- Експлицитно су пољима A1, A2 и A3 дефинисане сва три операнда (локације?)



- **Напомена**: Код неких процесора поља A1 и A2 дефинишу изворишне операнде, а A3 одредишни операнд. Међутим, могу и A2 и A3 бити изворишни, а A1 одредишни операнд.

Троадресни формат инструкције

- Треба срачунати израз
- A = B + C;

- Асемблерски код:
- ADD A, B, C

- A, B и C представљају симболичке ознаке појединих операнада. Детаљу и секције која се односи на начине адресирања.

Двоадресни формат инструкције

- Експлицитно су пољима A1 и A2 дефинисана два изворишна операнда.
- Одредиште се не дефинише експлицитно већ се користи или A1 или A2.

Двоадресни формат инструкције

- Треба срачунати израз
- C = A + B;

- Асемблерски код:
- MOV A, C
- ADD C, B

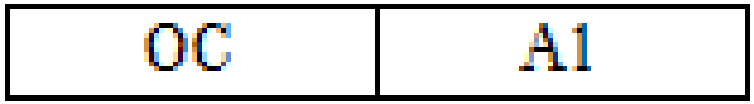
Двоадресни формат инструкције

- Добра страна двоадресног формата у односу на троадресни формат је да је инструкција краћа.

- Лоша страна формата је потреба за већим бројем инструкција.

Једноадресни формат инструкције

- Експлицитно се пољем A1 дефинише један изворишни операнд.



- Код ових процесора постоји посебан регистар процесора који се обично назива акумулатор који представља једно имплицитно изворište и имплицитно одредиште.

Једноадресни формат инструкције

- Треба срачунати израз
- C = A + B;

- Асемблерски код:
- LOAD A
- ADD B
- STORE C

Једноадресни формат инструкције

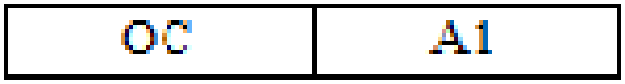
- Добра страна једноадресног формата у односу на двоадресни формат је да је инструкција краћа.
- Лоша страна је да је потребно још више инструкције да се реализује срачунавање истог израза.

Нулаадресни формат инструкције

- Постоји само поље кода операције и не постоји ни једно поље којим би се експлицитно дефинисале изворишне и одредишне операнде.
- Врх стека је имплицитно извориште за оба операнда и имплицитно одредиште за резултат.

*Изузетак су две инструкције са једним операндом:

- PUSH којом се операнд смешта на врх стека
- POP којом се операнд са врха стека смешта на одредиште



Нулаадресни формат инструкције

- Треба срачунати израз
- C = A + B;

- Асемблерски код:
- PUSH A
- PUSH B
- ADD
- POP C

Нулаадресни формат инструкције

- Добра страна нулаадресног формата у односу на једноадресног формат је да је инструкција краћа.
- Лоша страна је да је потребно још више инструкције да се реализује срачунавање истог израза.

- (Сви коментари су исти?)

Број операнада

- Уколико је процесор са неким форматом инструкција то не значи да све инструкције користе све операнде.
- Код неких инструкција или начина адресирања могуће је да су неки операнди вишак и та поља се онда не користе за ту намену.

Наредна инструкција?

- По завршетку извршавања текуће инструкције одакле узети следећу инструкције?
- Одређено је текућом вредношћу регистра PC.
- Код читања инструкције као адреса меморијске локације увек се користи текућа вредност регистра PC, која се том приликом инкрементира.
- Резултат => секвенцијално читати и извршавати.

Наредна инструкција?

- Постоји потреба за гранањем у програму.
- Не треба прећи на прву следеће инструкције у секвенци, већ на неку инструкцију ван секвенце.
- То се постиже инструкцијама скокова, које треба да се убаце у делове програма када треба одступити од секвенцијалног извршавања програма.
- Ефекат инструкција скокова је уписивање нове вредности у програмски бројач PC.

Могуће локације операнада

- Могуће локације операнада су
  - меморијске локације,
  - регистри процесора,
  - непосредне величине у инструкцији.
- Више варијанти реализације овог поља,а мотиви код његовог дефинисања су да:
  - овај део буде краћи да би се мање меморије заузимало за програме и брже очитавале инструкције (адреса меморије, непосредна величина или адреса регистра),
  - се брже долази до операнада (меморије, непосредна величина или регистар),
  - се пружи подршка за могућу примену неких техника реализације процесора (pipeline)

Врсте архитектура

- На основу могућих локација операнада разликују се три врсте архитектура:
  - меморија – меморија, (тро, дво, једно, и нулаадресни)
  - меморија – регистар, (двоадресни)
  - регистар – регистар. (троадресни)

Скуп инструкција

- Reduced Instruction Set Computers (RISCs)
  - Једноставне инструкције
  - Флексибилне
  - Већи проток
  - Брже извршавање
- Complex Instruction Set Computers (CISCs)
  - Хардверска подршка за програмске језике вишег нивоа
  - Компактни програми

RISC

- Само Load/Store инструкције приступају директно меморији
- Рад са подацима на нивоу трансфера регистар-регистар
- Једноставни начини адресирања
- Сви формати инструкција су исте дужине
- Инструкције раде јеноставне операције
- Обично једна инструкција по циклусу (једноставна инструкција)

CISC

- Већина инструкција има приступ меморији
- Велики број начина адресирања
- Инструкције имају различиту дужине
- Инструкције обављају и једноставне комплексне операције
- Више циклуса по инструкцији (комплексне инструкције)

Скуп инструкција

- Скуп стандардних инструкција чине:
  - аритметичке инструкције,
  - логичке инструкције,
  - инструкције померања и ротирања,
  - инструкције скока,
  - инструкције преноса,
  - мешовите инструкције

Аритметичке инструкције

- Аритметичким инструкцијама се реализују стандардне аритметичке операције попут сабирања, одузимања, множења, дељења, инкрементирања, декрементира, и поређења.
- Неки мнемоници би били:
  - ADD
  - SUB
  - MUL
  - DIV
  - INC
  - DEC
  - CMP

Аритметичке инструкције - ADD

- Троадресним форматом инструкција:
- ADD *a*, *b*, *c*

- ADD је симболички означено поље кода операције, а са *a*, *b* и *c* спецификације два изворишна (*b* и c) и једног одредишног операнда (a).

- *b* и *c* могу да буду било која комбинација регистара процесора, меморијских локација и непосредних величина у инструкцији.
- *a* не може да буде непосредна величина.

Аритметичке инструкције - ADD

- Најчешће комбинације:
  - *a*, *b* и *c* су регистри процесора,
  - *a* је регистар процесора, *b* је регистар процесора и *c je* непосредна величина,
  - *a*, *b* и c су меморијске локације,
  - итд.

Аритметичке инструкције - ADD

- Двоадресним форматом инструкција:
- ADD *a*, *b*

- ADD је симболички означено поље кода операције, а са *a*, *b* спецификације два изворишна операнда од којих је један и одредишни операнд (а у примеру).

- *b* може да буду регистар процесора, меморијска локација и непосредна величина.
- *a* не може да буде непосредна величина.

Аритметичке инструкције - ADD

- Најчешће комбинације:
  - *a* и *b* су регистри процесора,
  - *a* је регистар процесора, *b* је непосредна величина,
  - *a* и *b* су меморијске локације (ретко)
  - итд.

Аритметичке инструкције - ADD

- Једноадресним форматом инструкција:
- ADD *a*

- ADD је симболички означено поље кода операције, а са *a* спецификација другог изворишног операнда, при чему је акумулатор имплицитно изворište првог операнда и имплицитно одредиште резултат.

Аритметичке инструкције - ADD

- Најчешће комбинације:
  - *a* је регистар процесора
  - *a* је меморијска локација
  - *a* је непосредна величина
    (код неких инструкција није дозвољено)

Аритметичке инструкције - ADD

- Нулаадресним форматом инструкција:
- ADD

- ADD је симболички означено поље кода операције, при чему је врх стека имплицитно изворište за оба изворишна операнда и имплицитно одредиште.
- Током извршавања ове инструкције са врха стека се имплицитно чита најпре први а потом и други изворишни операнд, затим се над њима реализује операција сабирања и на крају се добијени резултат смешта на врх стека као имплицитно одредиште.

Аритметичке инструкције - ADD

- Слична је ситуација и са форматима инструкција за операције одузимања, множења и дељења.

Аритметичке инструкције - INC

- Посебан случај операције сабирања код које је имплицитно одређено да је вредност која се сабира једнака 1.
- Двоадресни формат:
- INC *a*, *b*
- INC је симболички означено поље кода операције, а са *a* и *b* спецификације изворишног (b) и одредишног операнда (a).

*Нема троадресни формат. Односно поље намењено за други изворишни операнд се не користи.

Аритметичке инструкције - INC

- Најчешће комбинације:
  - *a* и *b* су регистри процесора,
  - *a* је регистар процесора, *b* је непосредна величина,
  - *a* и *b* су меморијске локације,
  - итд.

Аритметичке инструкције - INC

- Једноадресним форматом инструкција:
- INC *a*

- INC је симболички означено поље кода операције, а са *a* спецификација изворишног и одредишног операнда.

* Некада је акумулатор имплицитно одредиште.

Аритметичке инструкције - INC

- Најчешће комбинације:
  - *a* је меморијска локација
  - *a* је регистар процесора,

Аритметичке инструкције - INC

- **Нулаадресним** форматом инструкција:
- INC

- INC је симболички означено поље кода операције, при чему је врх стека имплицитно изворište операнда и имплицитно одредиште.
- Током извршавања ове инструкције са врха стека се имплицитно чита најпре операнд, затим се над њим реализује операција увећавања за један и на крају се добијени резултат смешта на врх стека као имплицитно одредиште.

Аритметичке инструкције - INC

- **Безадресним** форматом инструкција:
- INC

- INC је симболички означено поље кода операције, при чему је акумулатор имплицитно изворište и имплицитно одредиште.
- Понекада се означава са INCA

Нулаадресни <> Безадресни

Аритметичке инструкције - INC

- Нулаадресни формат:
- INC; // скине са стека у акумулатор
  // увећа акумулатор за један
  // сними акумулатор на стек

- Безадресни формат:
- INC; // увећа акумулатор за један

Аритметичке инструкције - CMP

- Посебан случај операције одузимања код које се резултат одузимања никуда не уписује.
- Постављање индикатора N, Z, C и V у програмској статусној речи PSW процесора.
- Ова инструкција се назива аритметичко упоређивање.
- Формати инструкције упоређивања за процесоре са двоадресним, једноадресним и нулаадресним форматима инструкција су:
  - CMP *a*, *b*
  - CMP *a*
  - CMP

Аритметичке инструкције - CMP

- Индикатор N се поставља на вредност 1 уколико је резултат извршене операције одузимања негативан, док се у супротном случају поставља на вредност 0. Усвојено правило је да се индикатор N поставља на вредност најстаријег бита резултата операције одузимања.
- Индикатор Z се поставља на вредност 1 уколико је резултат извршене операције одузимања нула, док се у супротном случају поставља на вредност 0.
- Ови индикатори се поставља за све типове података.

Аритметичке инструкције - CMP

- Индикатор C се поставља на 1 уколико је први операнд мањи од другог операнда и представља позајмицу, док се у супротном случају поставља на 0.
- Индикатор V се поставља на вредност 1 уколико дошло до прекорачења (изласка из) опсега приликом операције одузимања.

Логичке инструкције

- Логичким инструкцијама се реализују стандардне логичке операције И, ИЛИ, ексклузивно ИЛИ и комплементирања.
- Мнемоници би били:
  - AND
  - OR
  - XOR
  - NOT

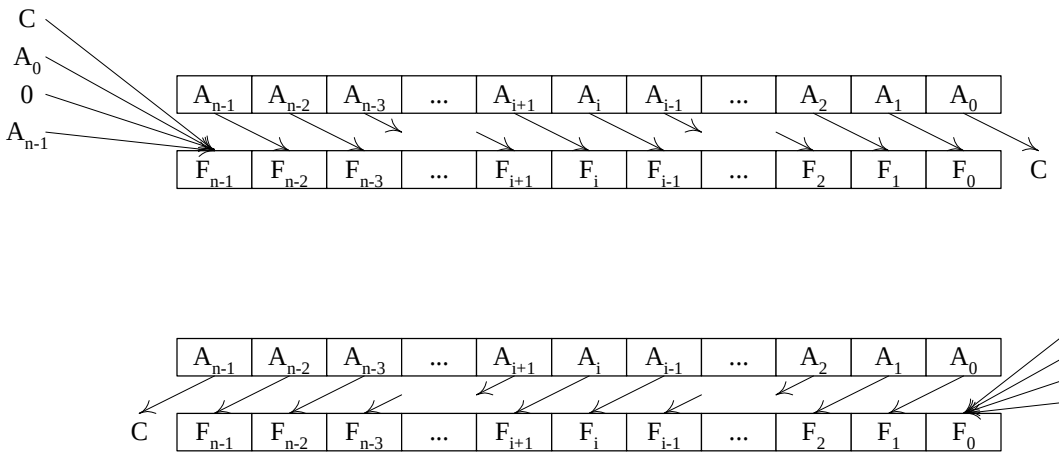
- Могу се јавити у свим форматима, осим инструкције NOT која нема троадресни формат.

Логичке инструкције - TST

- Посебан случај операције И код које се резултат никуда не уписује.
- Постављање индикатора N и Z у регистру PSW.
- Ова инструкција се назива логичко упоређивање.
- Формати инструкције упоређивања за процесоре са двоадресним, једноадресним и нулаадресним форматима инструкција су:
  - TST *a*, *b*
  - TST *a*
  - TST

Инструкције померања

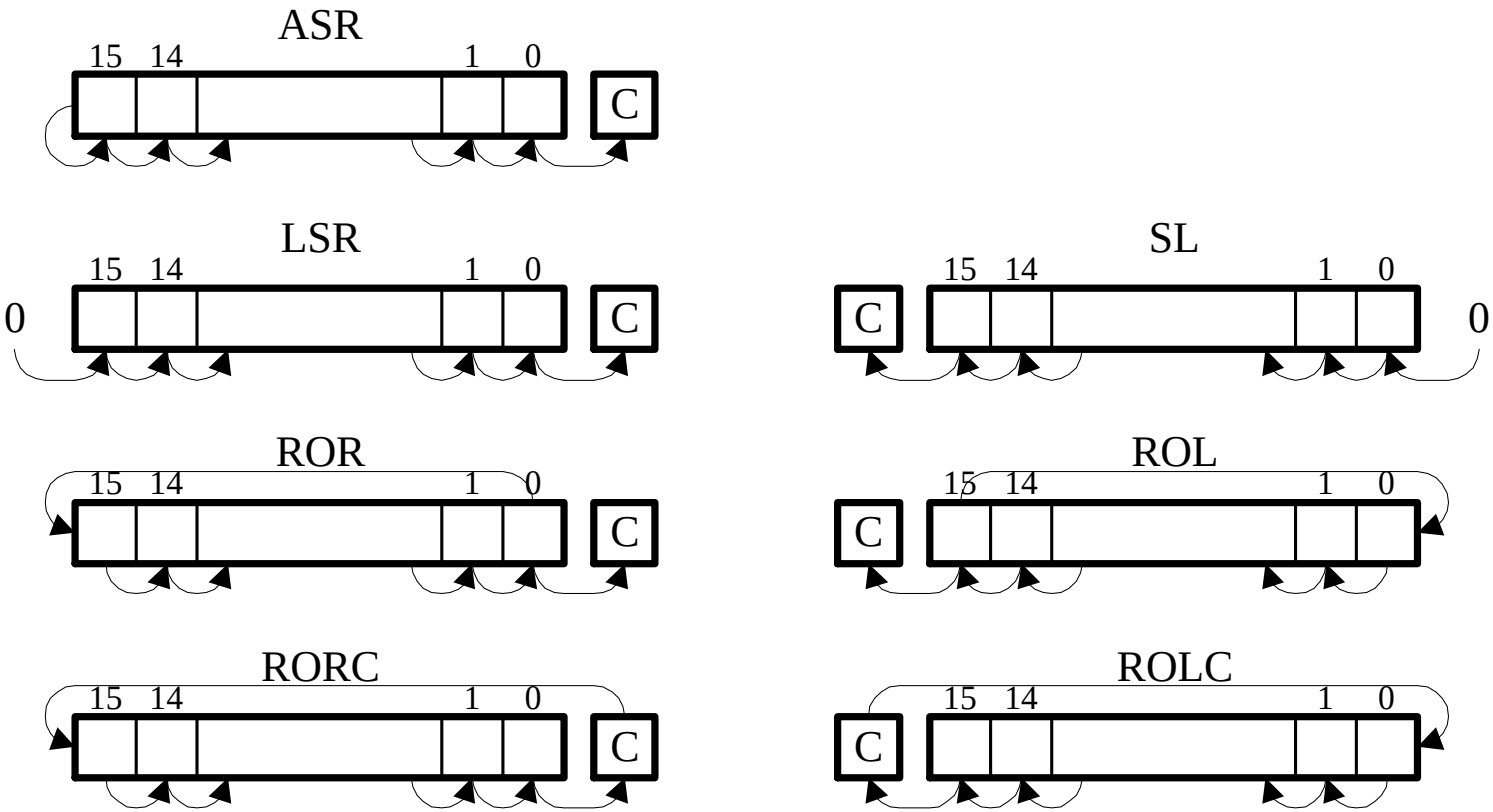
- Инструкцијама померања се реализују операције аритметичког и логичког померање удесно и улево, и ротација удесно и улево.
- Мнемоници би били:
  - ASR
  - ASL
  - LSR
  - LSL
  - ROR
  - ROL
  - (RORC)
  - (ROLC)



Инструкције померања - ASR

- Формати инструкције аритметичког померања удесно за процесоре са двоадресним, једноадресним и нулаадресним форматима инструкција су:
  - ASR *a*, *b*
  - ASR *a*
  - ASR
- Са ASR је симболички означено поље кода операције, а са *a* и *b* спецификације одредишних и изворишних операнада.

Померање за једно место



Инструкције преноса - MOV

- Двоадресни формат:
- MOV *a*, *b*
- MOV је симболички означено поље кода операције, а са *a* и *b* спецификације одредишног и изворишног операнда, респективно.

*Нема троадресни формат. Односно поље намењено за други изворишни операнд се не користи.

Инструкције преноса - MOV

- Најчешће комбинације:
  - *a* и *b* су меморијске локације
  - *a* и *b* су регистри процесора,
  - *a* је регистар процесора, *b* је непосредна величина,
  - итд.

Инструкције преноса - LOAD/STORE

- Једноадресни формат:
- LOAD *a*
- STORE *b*
- Са LOAD и STORE је симболички означено поље кода операције, а са *a* и *b* спецификације изворишног и одредишног операнда, респективно.

- Инструкцијом LOAD се операнд са изворишта *a* преноси у акумулатор као имплицитно одредиште,
- Инструкцијом STORE садржај акумулатора као имплицитно изворište преноси у одредиште *b*.

Инструкције преноса - LOAD/STORE

- Најчешће комбинације:
  - *b* није непосредна величина.

Инструкције преноса - IN/OUT

- Двоадресни формат:
- IN *regper*, *regproc*
- OUT *regproc*, *regper*
- Са IN је симболички означено поље кода операције за пренос из регистра контролера периферије у регистар процесора, при чему је са *regper* директно дата адреса регистра контролера периферије, док је са *regproc* директно дата адреса регистра процесора, са OUT је симболички означено поље кода операције за пренос из регистра процесора у регистар контролера периферије.

Инструкције преноса - IN/OUT

- Једноадресни формат:
- IN *regper*
- OUT *regper*
- Са IN је симболички означено поље кода операције за пренос из регистра контролера периферије у акумулатор, при чему је са *regper* директно дата адреса регистра контролера периферије, са OUT је симболички означено поље кода операције за пренос из акумулатора у регистар контролера периферије.

Инструкције преноса - PUSH/POP

- Нулаадресни формат:
- PUSH *a*
- POP *b*
- Са PUSH и POP је симболички означено поље кода операције, а са *a* и *b* спецификације изворишног и одредишног операнда, респективно.

- Инструкцијом PUSH се операнд са изворишта *a* преноси на стек као имплицитно одредиште,
- Инструкцијом POP садржај са стека као имплицитно изворište преноси у одредиште *b*.

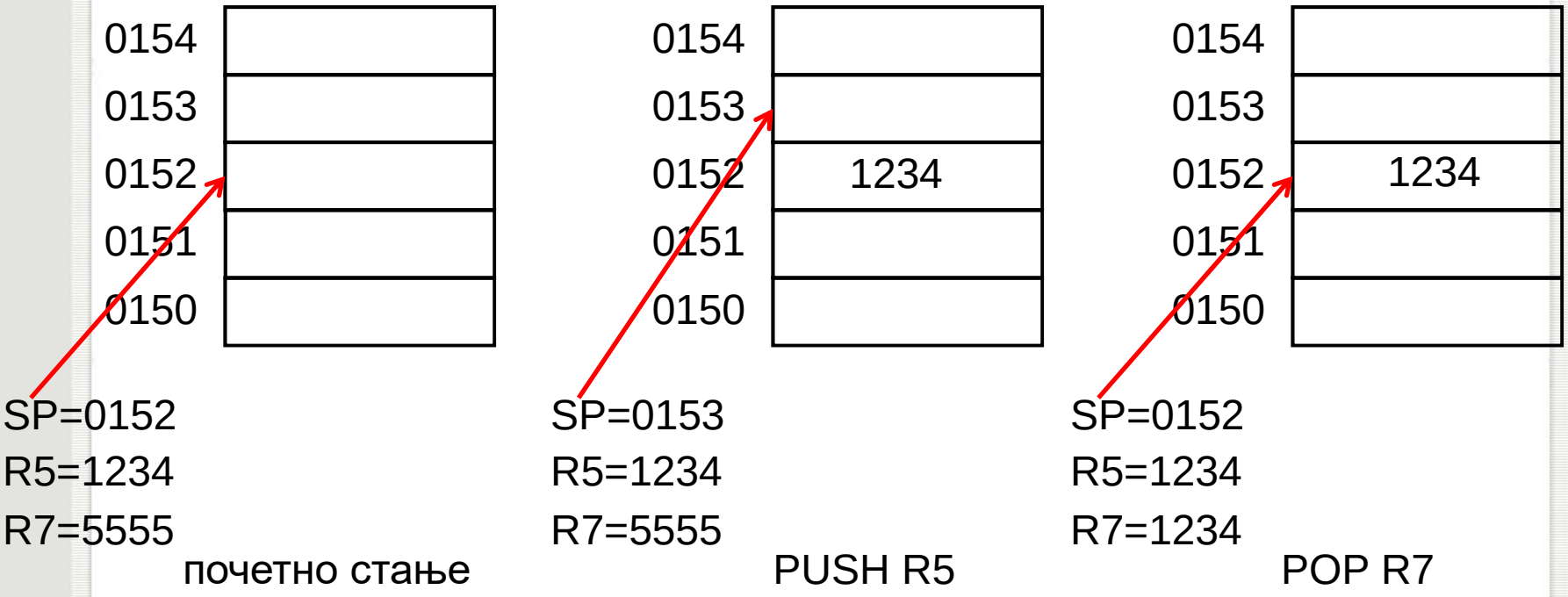
Инструкције преноса - PUSH/POP

- Најчешће комбинације:
  - *b* није непосредна величина.

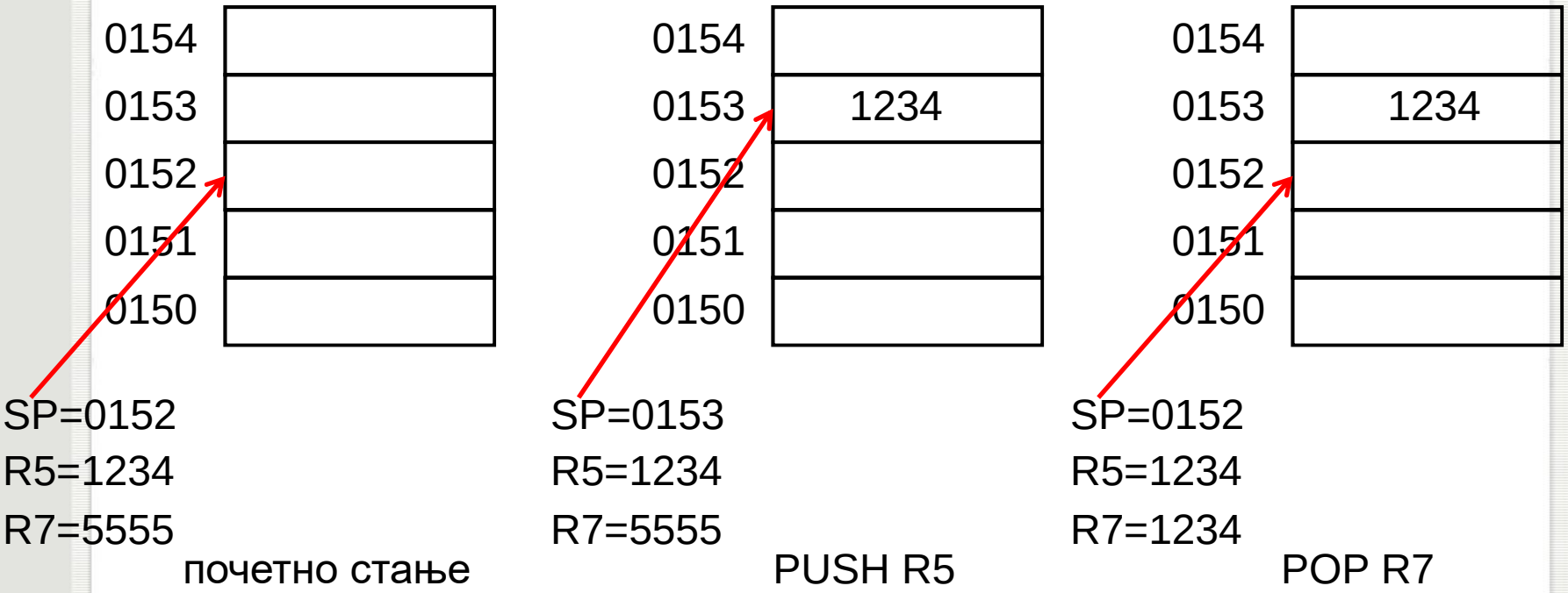
Варијанте раста стека

- Стек може да расте (да се попуњава) ка:
  - Нижим меморијским локацијама
  - Вишим меморијским локацијама
- Показивач на врх стека може да указује на:
  - Прву слободну локацију
  - Попуњену попуњену локацију

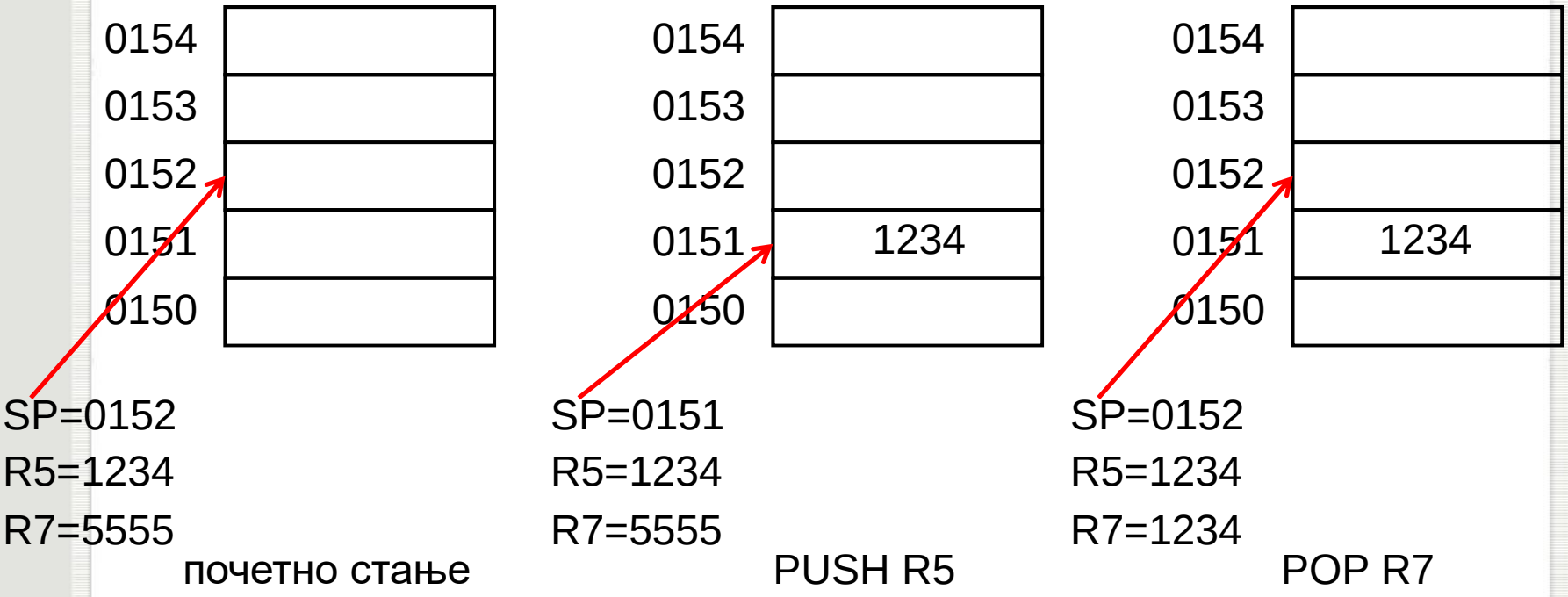
Варијанта: навише, на слободну



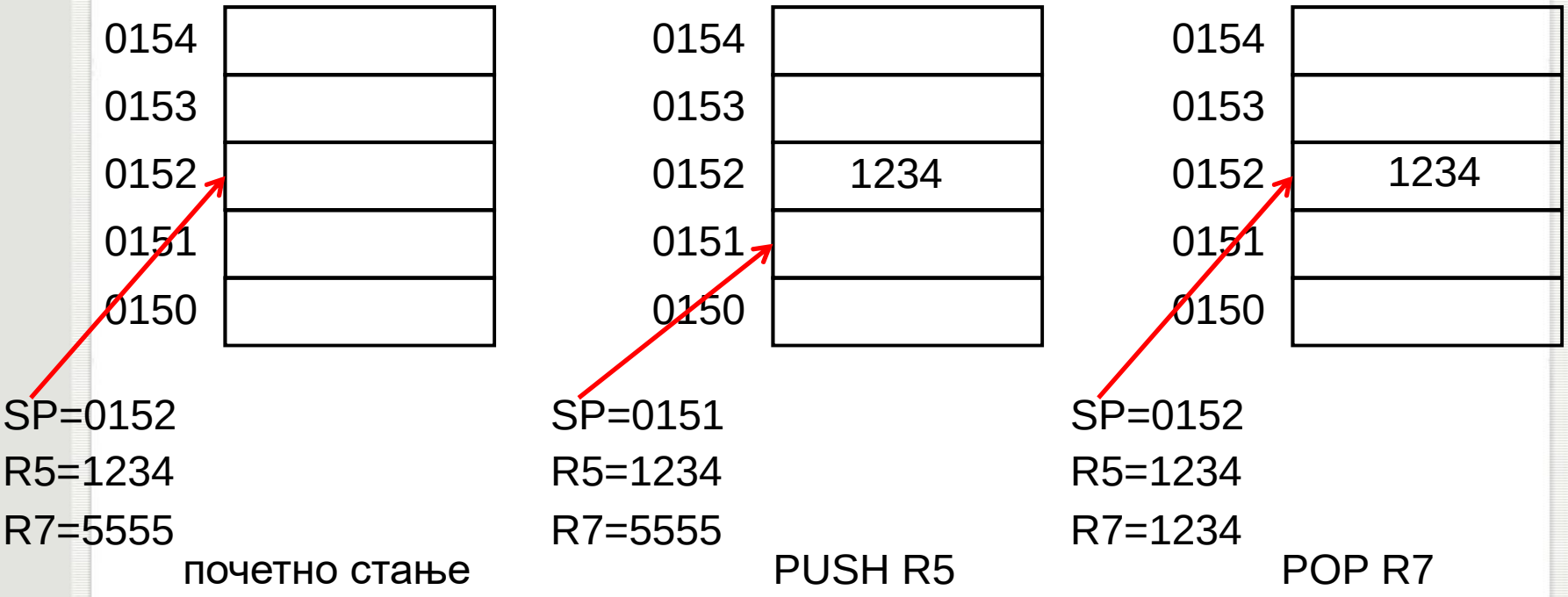
Варијанта: навише, на заузету



Варијанта: наниже, на заузету



Варијанта: наниже, на слободну



Инструкције скока

- Инструкције скока се сврставају у следеће групе:
  - инструкције безусловног скока,
  - инструкције условног скока,
  - инструкције скока на потпрограм,
  - повратка из потпрограма,
  - инструкција скока у прекидну рутину,
  - инструкција повратка из прекидне рутине.

Инструкција безусловног скока

- Формат инструкције безусловног скока је
- JMP *adresa*
- при чему је са JMP симболички означено поље кода операције, а са *adresa* поље са вредношћу адресе меморијске локације на којој се налази инструкција на чије извршавање треба прећи. Извршавања инструкције JMP се састоји у уписивању вредност из поља *adresa* у програмски бројач PC.

Инструкција безусловног скока

- У случају када адреса инструкције на коју треба скочити није позната у време превођења, већ се израчунава током извршавања програма, користи се инструкција безусловног скока чији је формат
- JMPIND *a*
- при чему је са JMPIND симболички означено поље кода операције, а са *a* поље са спецификацијом адресе меморијске локације на којој се налази инструкција на чије извршавање треба прећи.

Инструкција условног скока

- Формат инструкције условног скока је
- CODE *disp*
- при чему је са CODE симболички означено поље кода једне од операције безусловног скока, а са *disp* поље са вредношћу помераја са којим треба направити релативан скок у односу на текућу вредност програмског бројача уколико је услов за скок испуњен.

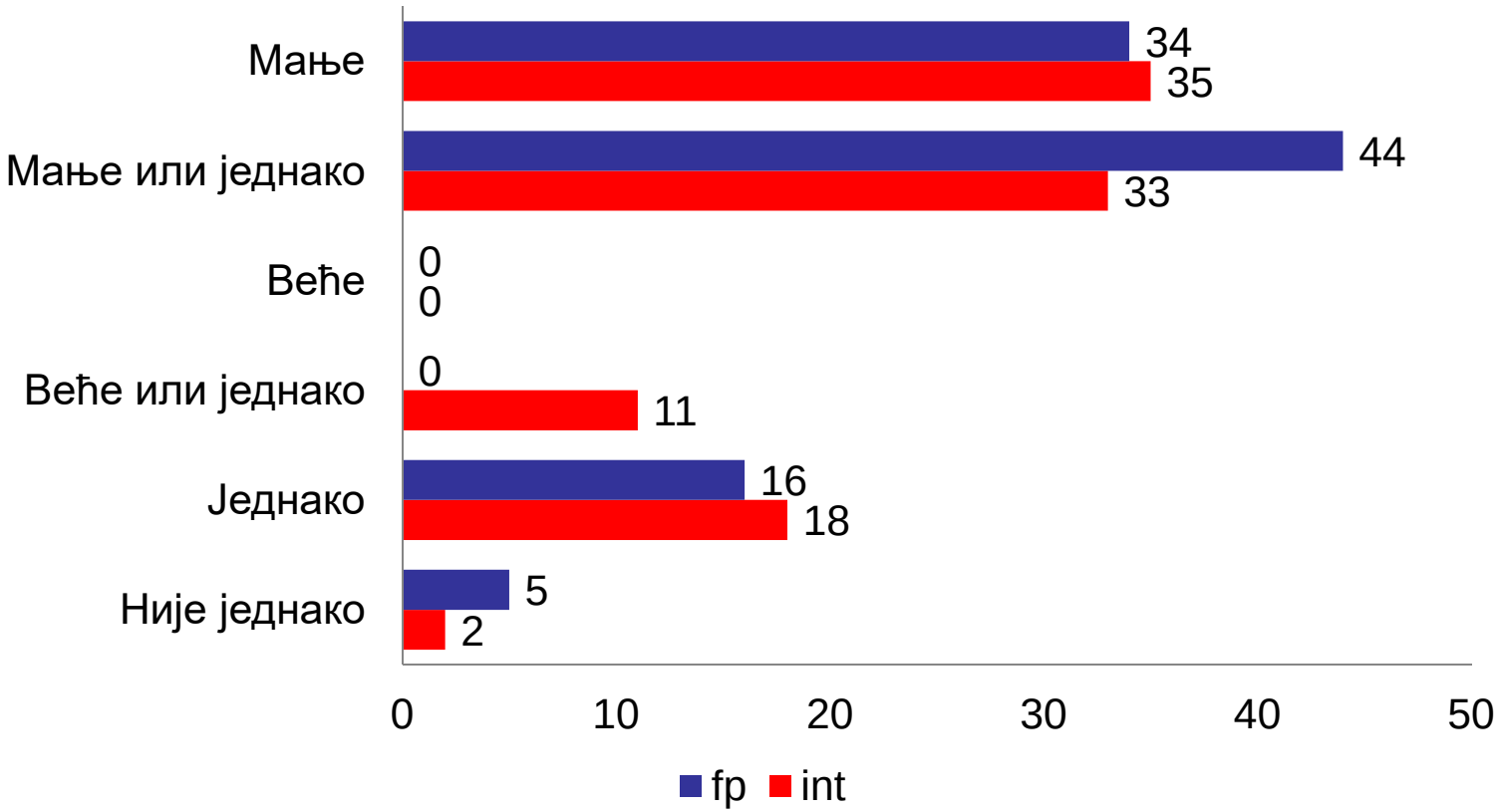
Инструкције условног скока

инструкција	значење	услов
BEQL	скок на једнако	Z = 1
BNEQ	скок на неједнако	Z = 0
BGRTU	скок на веће него (без знака)	C ∨ Z = 0
BGREU	скок на веће него или једнако (без знака)	C = 0
BLSSU	скок на мање него (без знака)	C = 1
BLEQU	скок на мање него или једнако (без знака)	C ∨ Z = 1
BGRT	скок на веће него (са знаком)	(N ⊕ V) ∨ Z = 0
BGRE	скок на веће него или једнако (са знаком)	N ⊕ V = 0
BLCC	скок на мање него (са знаком)	(N ⊕ V) = 1
BLEQ	скок на мање него или једнако (са знаком)	(N ⊕ V) ∨ Z = 1
BNEG	скок на N = 1	N = 1
BNNG	скок на N = 0	N = 0
BOVF	скок на V = 1	V = 1
BNVF	скок на V = 0	V = 0

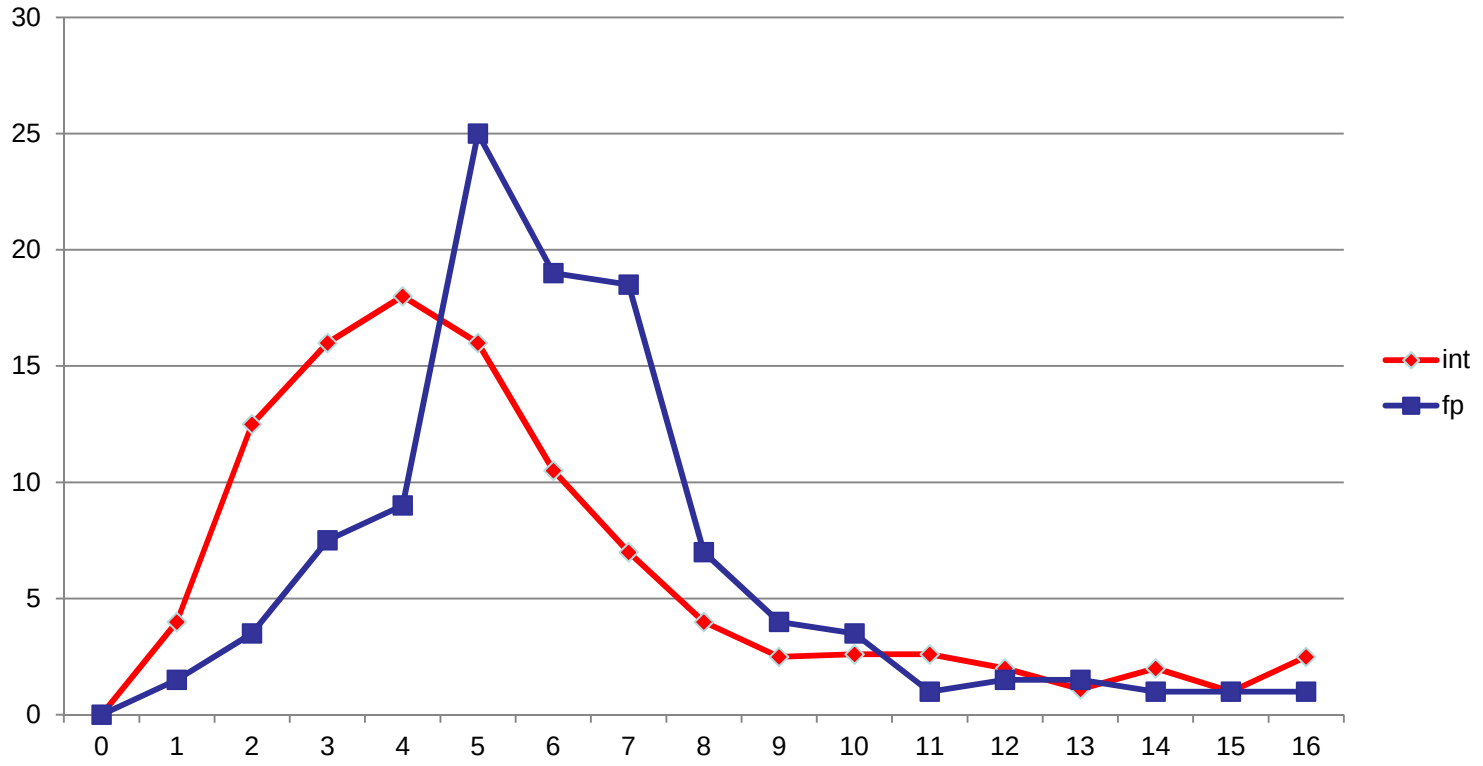
Инструкције условног скока - алтернатива

инструкција	значење	услов
BZ	скок на једнако	Z = 1
BNZ	скок на неједнако	Z = 0
BC	скок на мање него (без знака)	C = 1
BNC	скок на веће него или једнако (без знака)	C = 0
BN	скок на N = 1	N = 1
BNN	скок на N = 0	N = 0
BV	скок на V = 1	V = 1
BNV	скок на V = 0	V = 0

Инструкције условног скока



Дужина поља disp



- Број бита потребних за померај

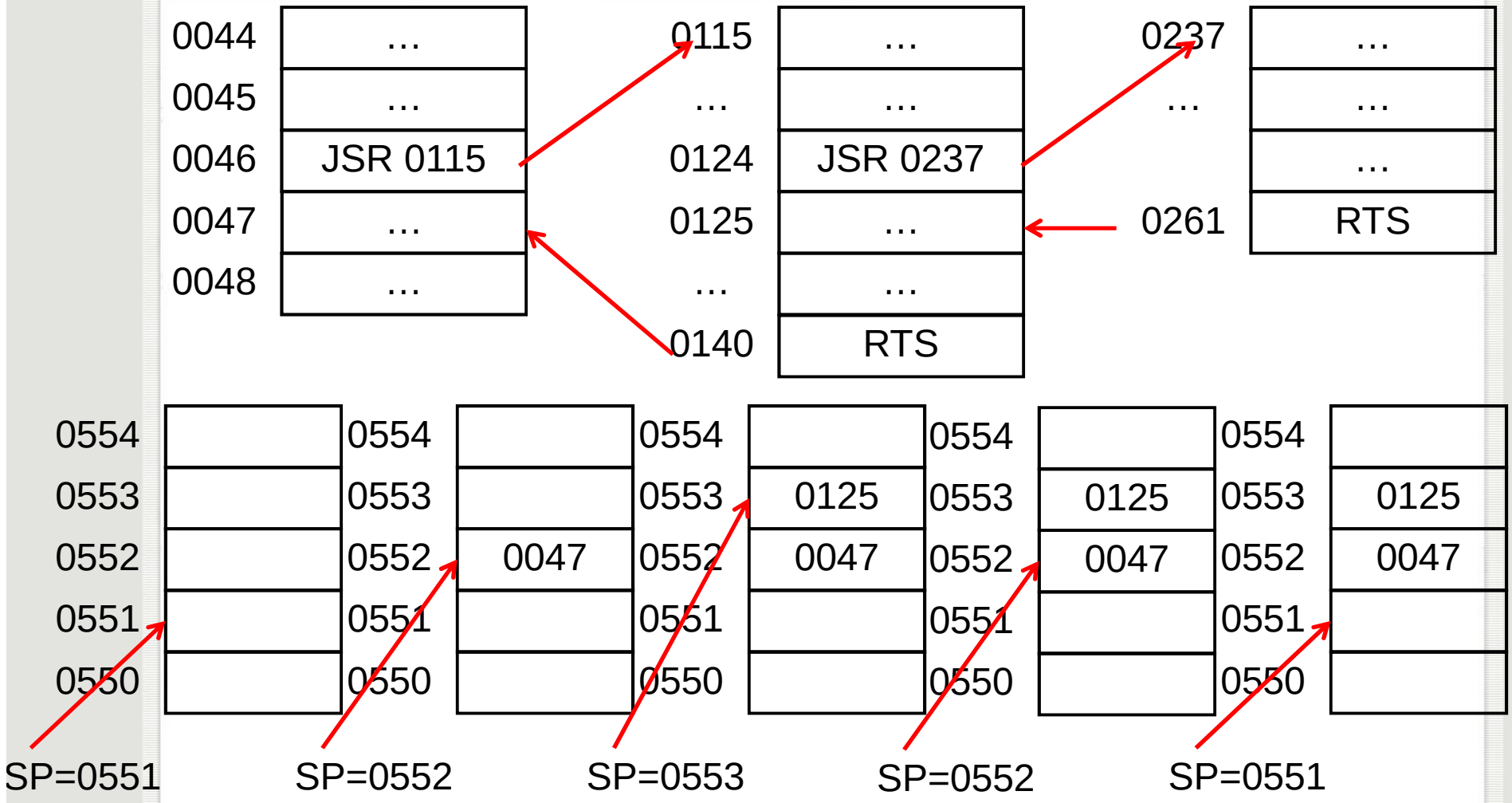
Инструкције скока на потпрограм

- Формат инструкције скока на потпрограм је
- JSR *adresa*
- при чему је са JSR симболички означено поље кода операције, а са *adresa* поље са вредношћу адресе меморијске локације на којој се налази прва инструкција потпрограма на чије извршавање треба прећи.
- Извршавање инструкције JSR се састоји у стављању на стек текуће вредности програмског бројача PC и уписивању вредности из поља *adresa* у програмски бројач PC.

Инструкције повратка из потпрограма

- Формат инструкције повратка из потпрограма је
- RTS
- при чему је са RTS симболички означено поље кода операције. Инструкција RTS мора да буде задња инструкција потпрограма.
- Извршавање инструкције RTS се састоји у скидању вредности са стека и уписивању у програмски бројач PC.

Скок на потпрограм



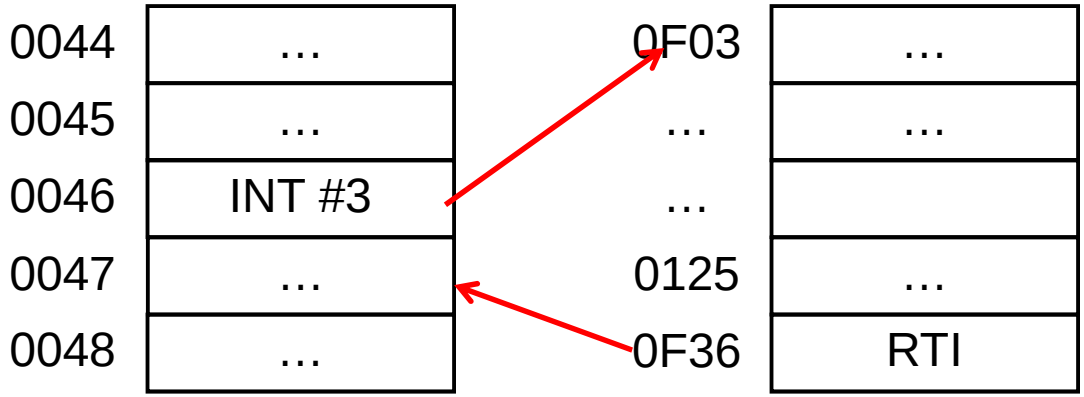
Инструкција скока на прекидну рутину

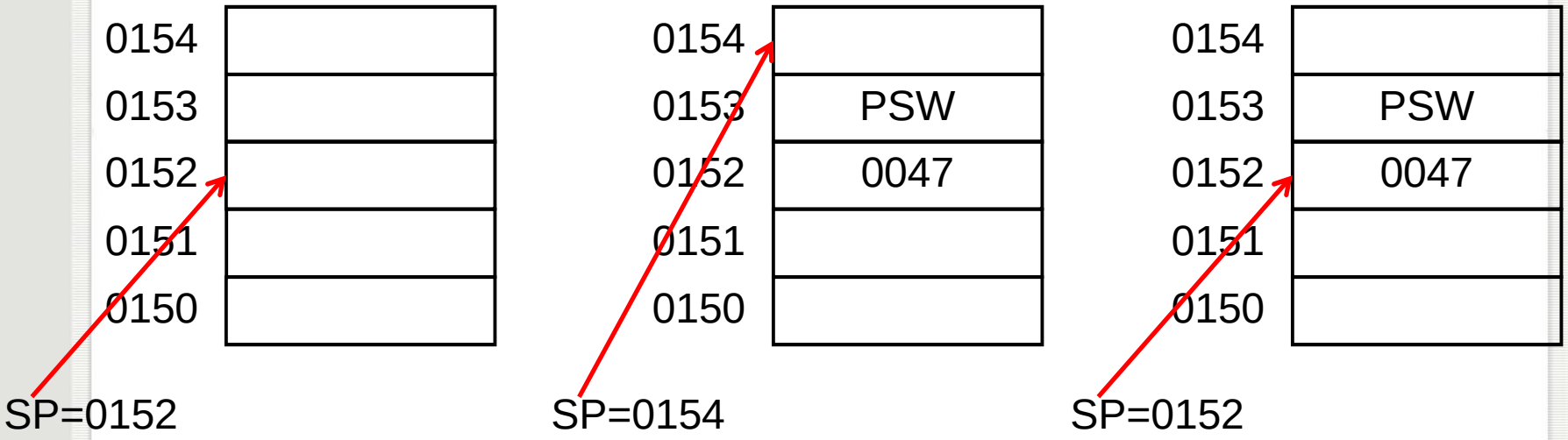
- Формат инструкције скока на прекидну рутину је
- INT *num*
- при чему је са INT симболички означено поље кода операције, а са *num* поље са вредношћу адресе меморијске локације на којој се налази прва инструкција прекидне рутине.
- Извршавање инструкције INT се састоји у стављању на стек текуће вредности програмског бројача PC, програмске статусне речи PSW, а у неким реализацијама процесора и свих програмски доступних регистара, и уписивању вредности из поља *adresa* у програмски бројач PC.

Инструкција повратка из прекидне рутине

- Формат инструкције повратка из повратка из прекидне рутине је
- RTI
- при чему је са RTI симболички означено поље кода операције. Инструкција RTI мора да буде задња инструкција прекидне рутине. Извршавање инструкције RTI се састоји у скидању вредности са стека и уписивању у програмски доступне регистре, уколико се ради о реализацијама процесора код којих се приликом скока на прекидну рутину ови регистри чувају на стеку, програмску статусну реч и програмски бројач PC.

Скок на прекид





Мешовите инструкције

- У скупу инструкција могу да се нађу и инструкције којима се омогућава задавање различитих режима рада рачунара, пружа подршка за тестирање програма итд.
- Пример:
  - INTE
  - INTD
  - TRPE
  - TRPD
  - NOP
  - итд.

Мешовите инструкције - INTE

- Формат инструкције задавања режима рада у коме се реагује на захтеве за прекид је
- INTE
- при чему је са INTE симболички означено поље кода операције. Извршавање инструкције INTE се састоји у уписивању вредности 1 у разред I регистра PSW.

Мешовите инструкције - INTD

- Формат инструкције задавања режима рада у коме се не реагује на захтеве за прекид је
- INTD
- при чему је са INTD симболички означено поље кода операције. Извршавање инструкције INTD се састоји у уписивању вредности 0 у разред I регистра PSW.

Мешовите инструкције - TRPE

- Формат инструкције задавања режима *прекид после сваке инструкције* је
- TRPE
- при чему је са TRPE симболички означено поље кода операције. Извршавање инструкције TRPE се састоји у уписивању вредности 1 у разред T регистра PSW.

Мешовите инструкције - TRPD

- Формат инструкције заустављање режима *прекид после сваке инструкције* је
- TRPD
- при чему је са TRPD симболички означено поље кода операције. Извршавање инструкције TRPDE се састоји у уписивању вредности 0 у разред T регистра PSW.

Мешовите инструкције

- Формат инструкције без дејства је
- NOP
- при чему је са NOP симболички означено поље кода операције. Извршавање инструкције NOP се састоји само у читању инструкције и инкрементирању програмског бројача PC, после чега се прелази на извршавање следеће инструкције.

Вероватноћа појединих инструкција

Ранг	Инструкција	Проценат
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move reg-reg	4%
9	call	1%
10	return	1%
Торал		96%

8086 инструкције

Програмски бројач PC

- Регистар чија се вредност имплицитно користи као адреса меморијске локације са које се чита инструкција.
- Вредност програмског бројача PC се мења:
  - имплицитно приликом сваког читања инструкције када се врши инкрементирање програмског бројача PC
  - експлицитно инструкцијама скока када се у програмски бројач бројач PC уписује нова вредност.

Регистри података DR

- Регистри података се уводе да би се убрзао приступ подацима, тако што би се током извршавања програма приступало подацима у регистрима података процесора уместо у меморијским локацијама, а резултат је чињенице да је приступ регистрима скоро за ред величине бржи од приступа меморијским локацијама.
- Регистрима података се приступа програмским путем инструкцијама преноса и аритметичким, логичким и померачким инструкцијама.

Адресни регистри AR

- Садржај адресног регистра користи:
  - као адреса меморијске локације са које се чита или у коју се уписује податак;
  - инструкцијом скока JMPIND када се садржај адресног регистра користи као адреса скока коју треба уписати у програмски бројач PC.
- Адресни регистри се уводе да би се убрзао приступ адресама, тако што би се током извршавања програма приступало адресама у адресним регистрима процесора уместо у меморијским локацијама, а резултат је чињенице да је приступ регистрима скоро за ред величине бржи од приступа меморијским локацијама.

Базни регистри BR

- Збир садржаја специфицираног базног регистра и помераја код базног адресирања, односно базног регистра, индексног регистра и помераја код базно-индексног адресирања, представља адресу меморијске локације на којој се налази изворишни или одредишни операнд.

- Користи се код базног и базно-индексног адресирања

Индексни регистри XR

- Збир садржаја специфицираног индексног регистра и помераја код индексног адресирања, односно базног регистра, индексног регистра и помераја код базно-индексног адресирања, представља адресу меморијске локације на којој се налази изворишни или одредишни операнд.
- Користе се код индексног и базно-индексног адресирања.

Регистри опште намене GPR

- Користе на исти начин као регистри података, адресни регистри, базни регистри и индексни регистри.
- Регистри опште намене GPR се јављају код оних процесора код којих не постоје посебни регистри података, адресни регистри, базни регистри и индексни регистри.
- Могу да користе било који од регистара уз било које адресирање.

Регистар PSW

- **С**тандардна програмска статусна реч процесора састављена од одређеног броја битова, који се обично називају индикатори.
- Битови програмске статусне речи PSW се независно постављају и користе по правилима дефинисаним посебно за сваки бит.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I	-	-	-	-	-	-	-	-	-	-	-	V	C	Z	N

Пример: Структура регистра PSW

Битови регистра PSW

- N—бит који се поставља на 1 у случају да је резултат операције негативан,
- Z—бит који се поставља на 1 у случају да је резултат операције једнак 0,
- C—бит који се поставља на 1 у случају преноса/позајмице у аритметици целобројних величина без знака *
- V—бит који се поставља на 1 у случају прекорачења у аритметици целобројних величина са знаком *
- I—бит који је једнак 1 ако треба да буду дозвољени маскирајући прекиди.

Битови регистра PSW

*Ради једноставније и униформније реализације код неких процесора бити C и V се код аритметичких инструкција **увек** постављају и то на основу излаза ALU јединице.

- Бит C на највиши бит преноса C_n.
  $C = C_n$
- Бит V на разлику највиших бит преноса C_n и C_{n-1}.
  $V = C_n$ XOR C_{n-1}

Регистар SP

- Указивач на врх стека када је стек је организован у оперативној меморији.
- Подршка за реализацију LIFO структуре података.
- регистра SP се користи као адреса меморијске локације у коју треба уписати податак или из које треба очитати податак.
- Ажурирање и то инкрементирање или декрементирање садржаја регистра SP.

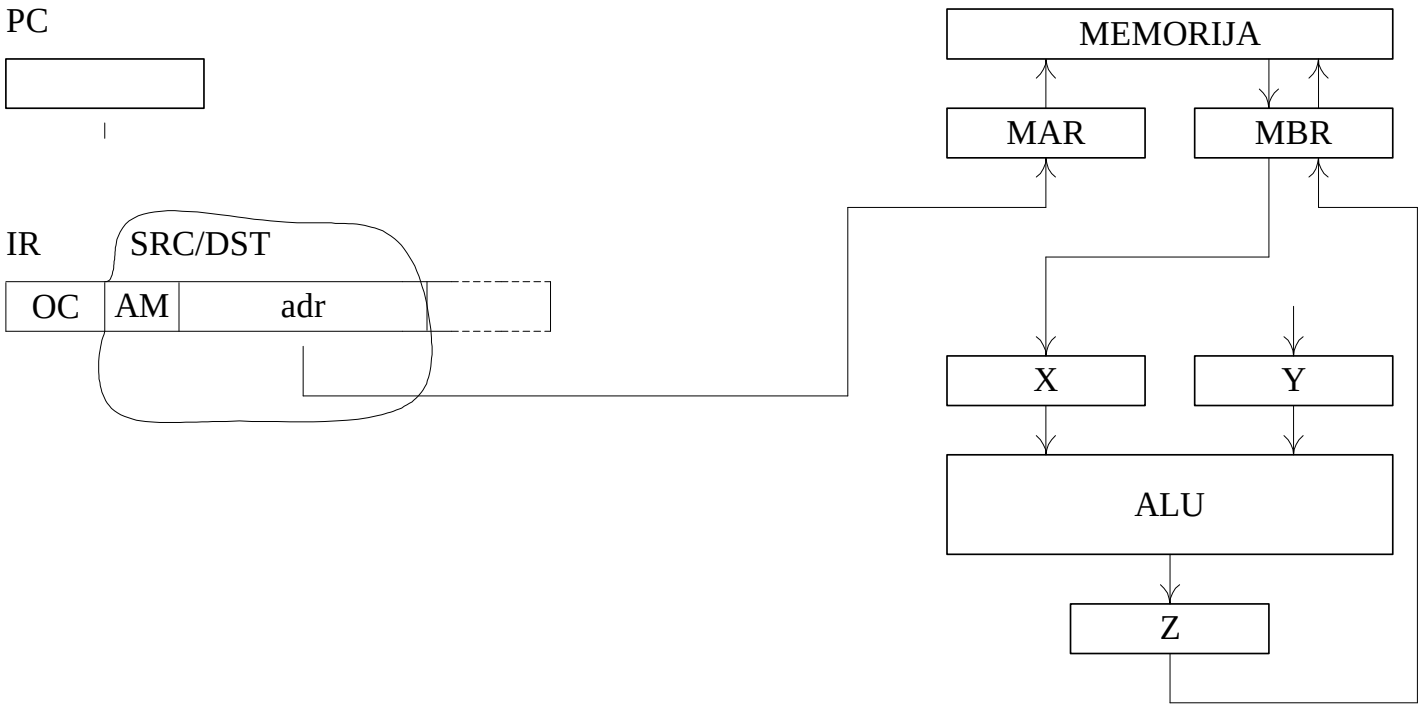
Начини адресирања

- Начини адресирања одређују да ли је операнд садржај неке меморијске локације, неког од регистара података или регистара опште намене процесора или непосредна величина у самој инструкцији.
- Начини адресирања специфицирају и како треба формирати адресу меморијске локације уколико је операнд садржај неке меморијске локације.

Меморијско директно адресирање

- Меморијско директно адресирање је адресирање код кога се операнд налази у меморијској локацији, а у адресном делу инструкције у коме се специфицира операнд, поред групе битова којима се задаје начин адресирања (AM), постоји и група битова којима се задаје адреса меморијске локације (adr).

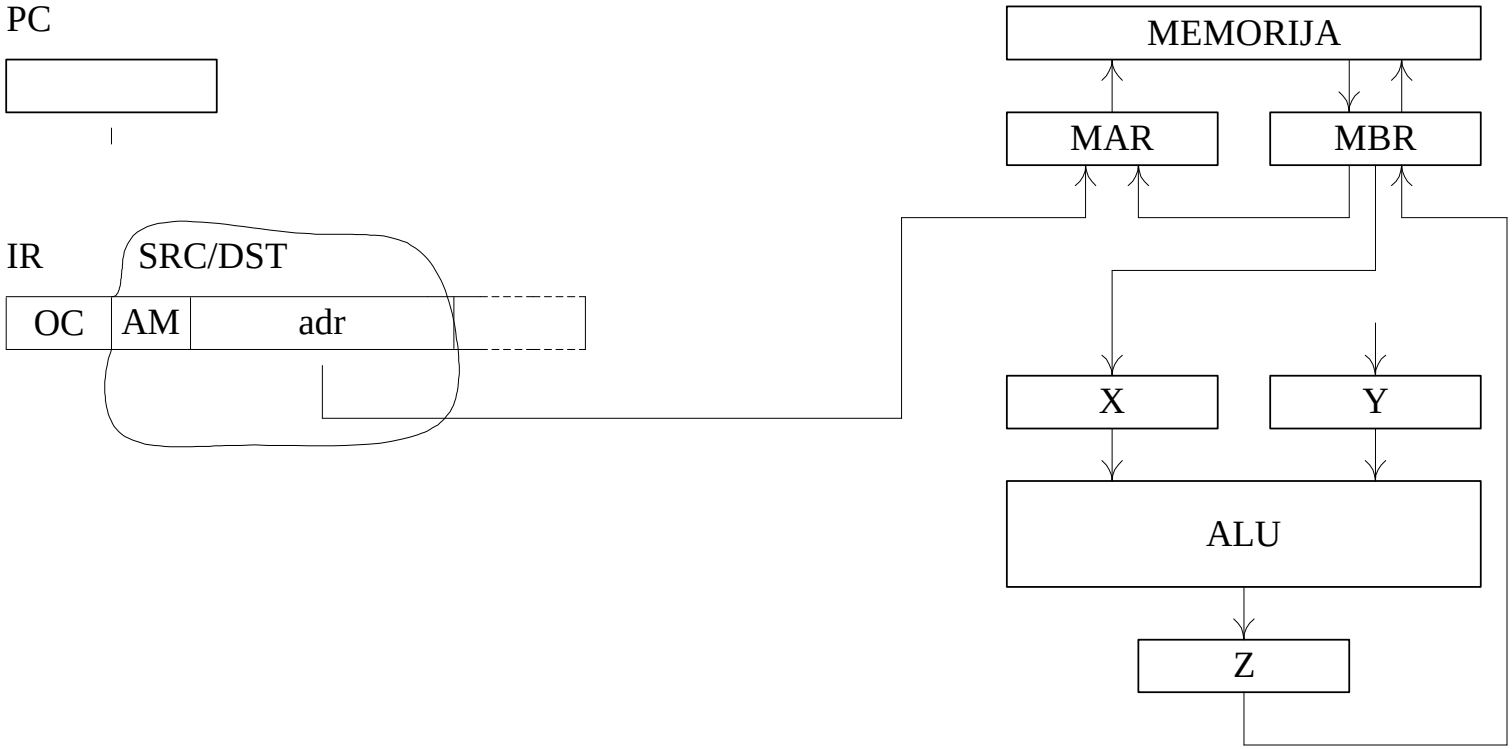
Меморијско директно адресирање



Меморијско индиректно адресирање

- Меморијско индиректно адресирање је адресирање код кога се операнд налази у меморији, а у адресном делу инструкције у коме се специфицира операнд, поред групе битова којима се задаје начин адресирања (AM), постоји и група битова којима се задаје адреса меморијске локације на којој се налази адреса операнда.

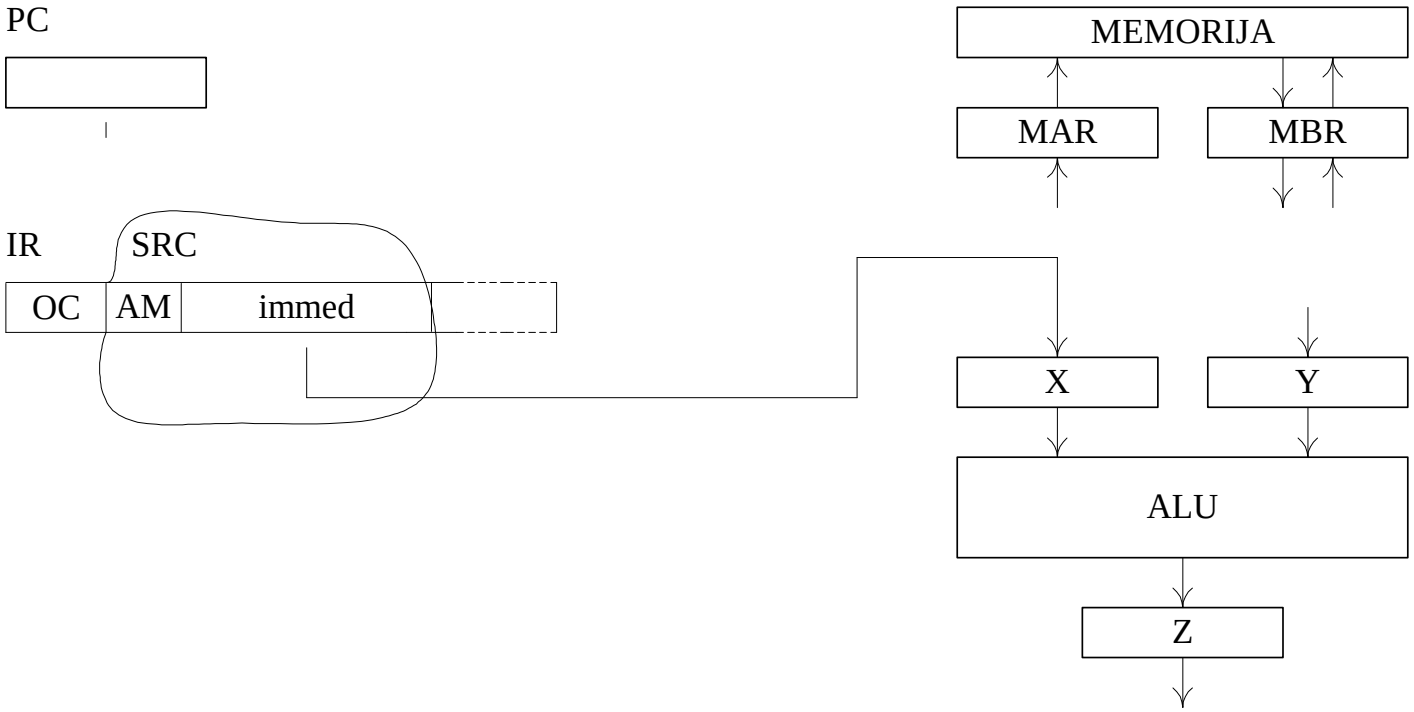
Меморијско индиректно адресирање



Непосредно адресирање

- Непосредно адресирање је адресирање код кога се операнд налази непосредно у самој инструкцији.

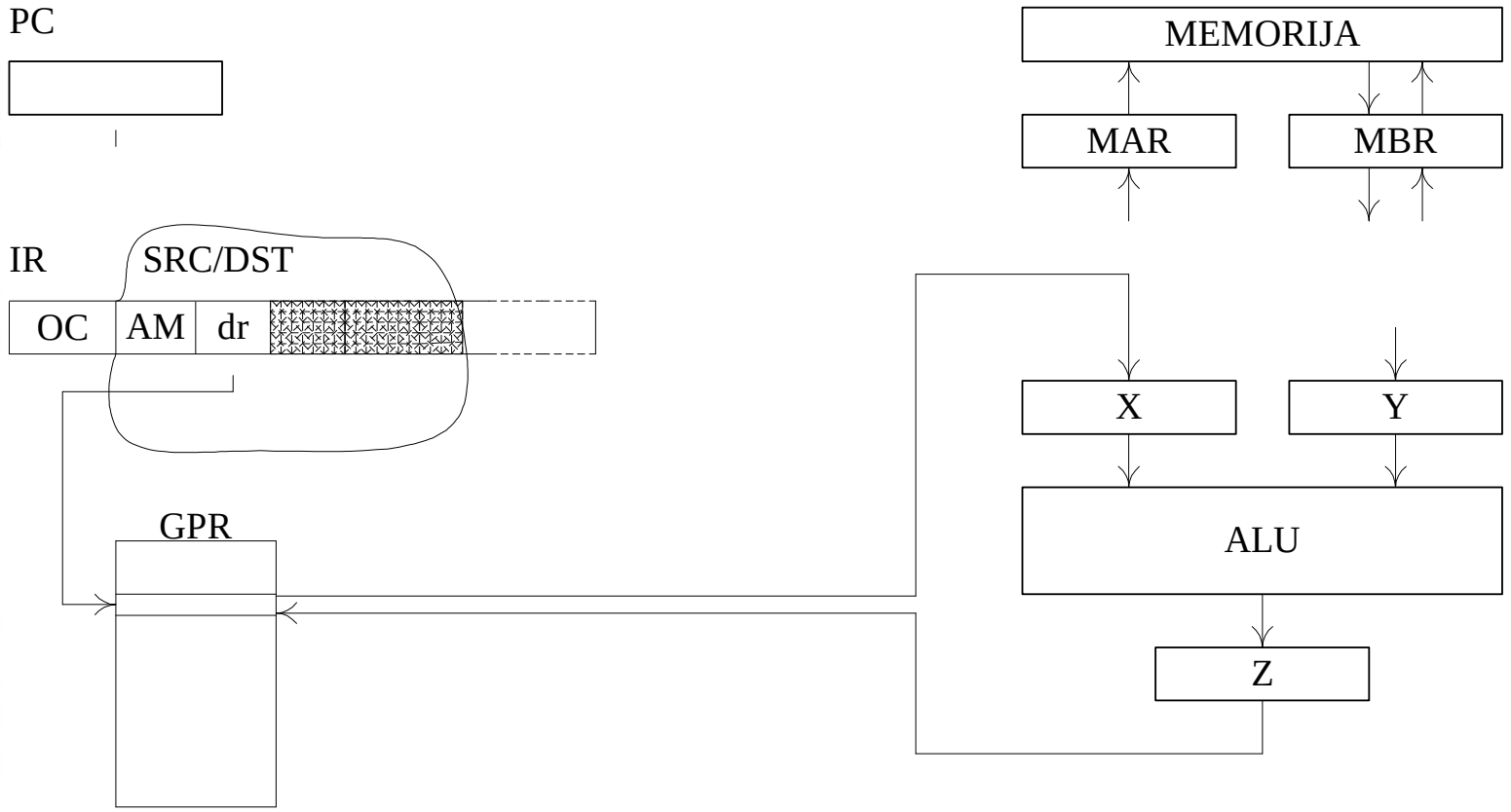
Непосредно адресирање



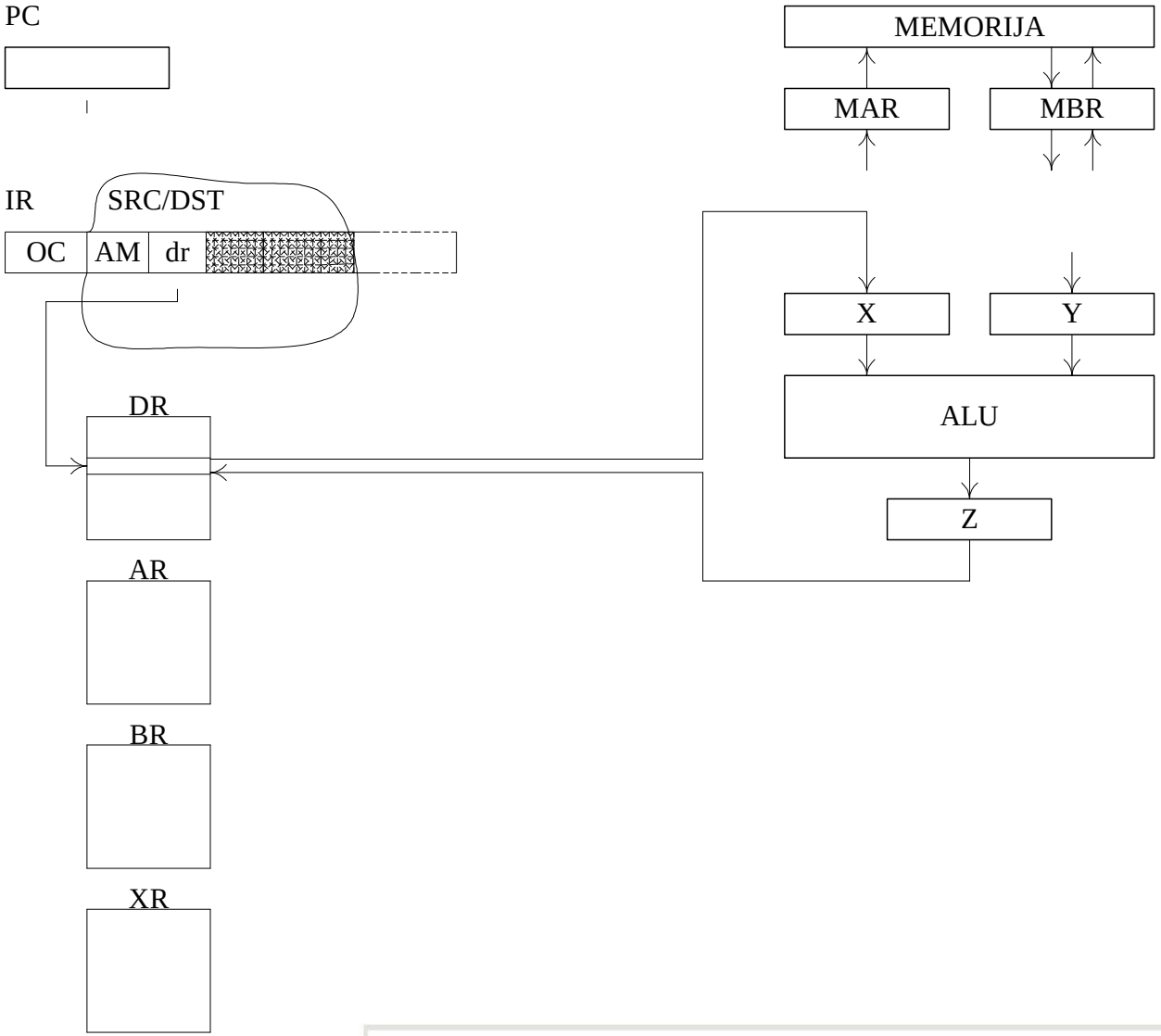
Регистарско директно адресирање

- Регистарско директно адресирање је адресирање код кога се операнд налази у једном од регистара података или регистара опште намене процесора.
- Код оних процесора код којих постоје посебно регистри података (DR), адресни регистри (AR), базни регистри (BR) и индексни регистри (XR), операнд се налази у једном од регистара података (DR), док код оних процесора код којих постоје само регистри опште намене (GPR), операнд се налази у једном од регистара опште намене.

Регистарско директно адресирање



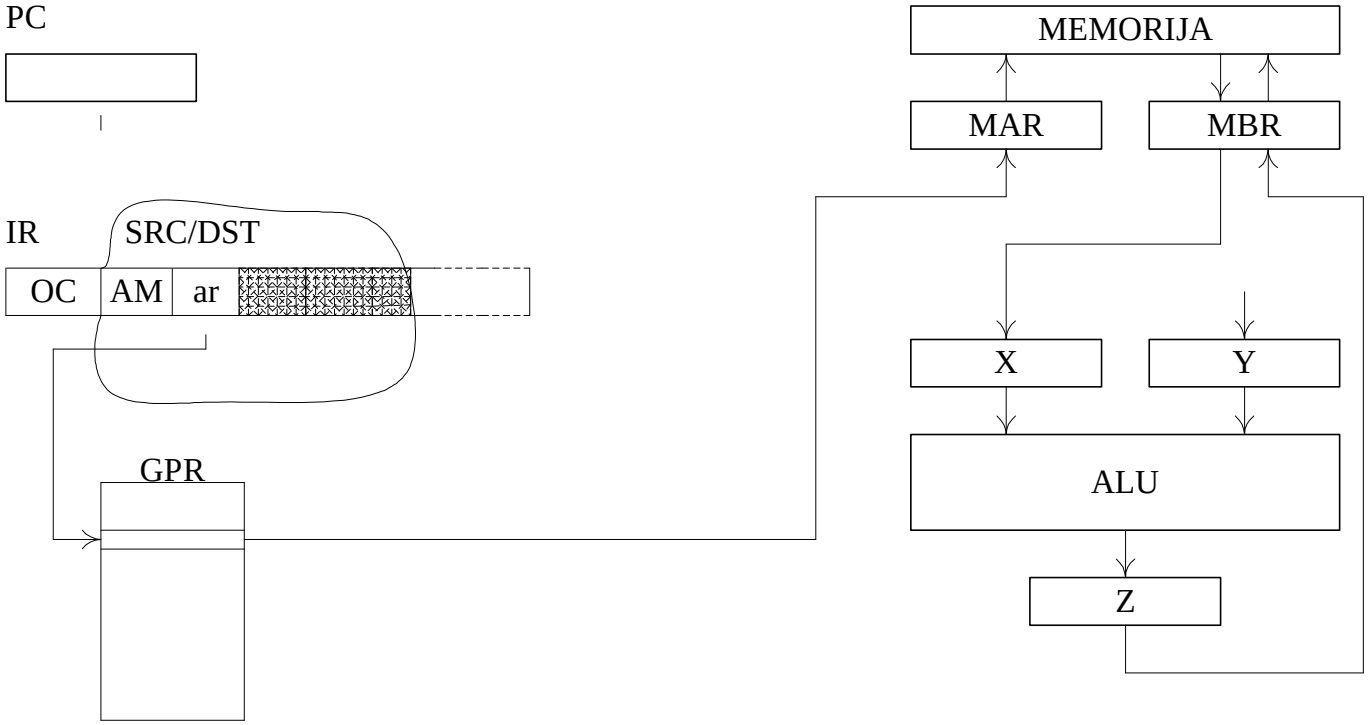
Регистарско директно адресирање



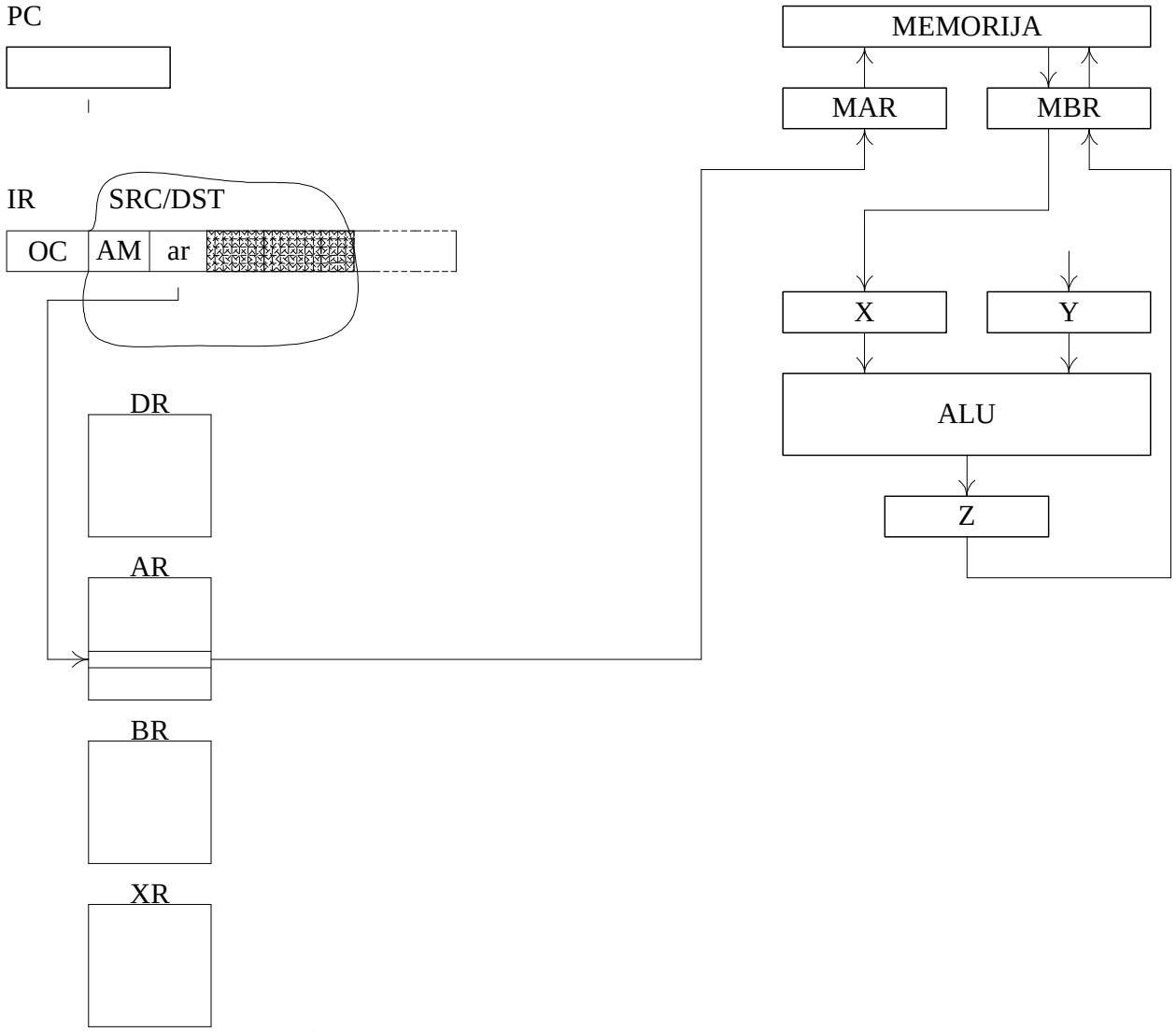
Регистарско индиректно адресирање

- Регистарско индиректно адресирање је адресирање код кога се операнд налази у меморијској локација, а адреса меморијске локације се налази у једном од адресних регистара или регистара опште намене процесора.

Регистарско индиректно адресирање



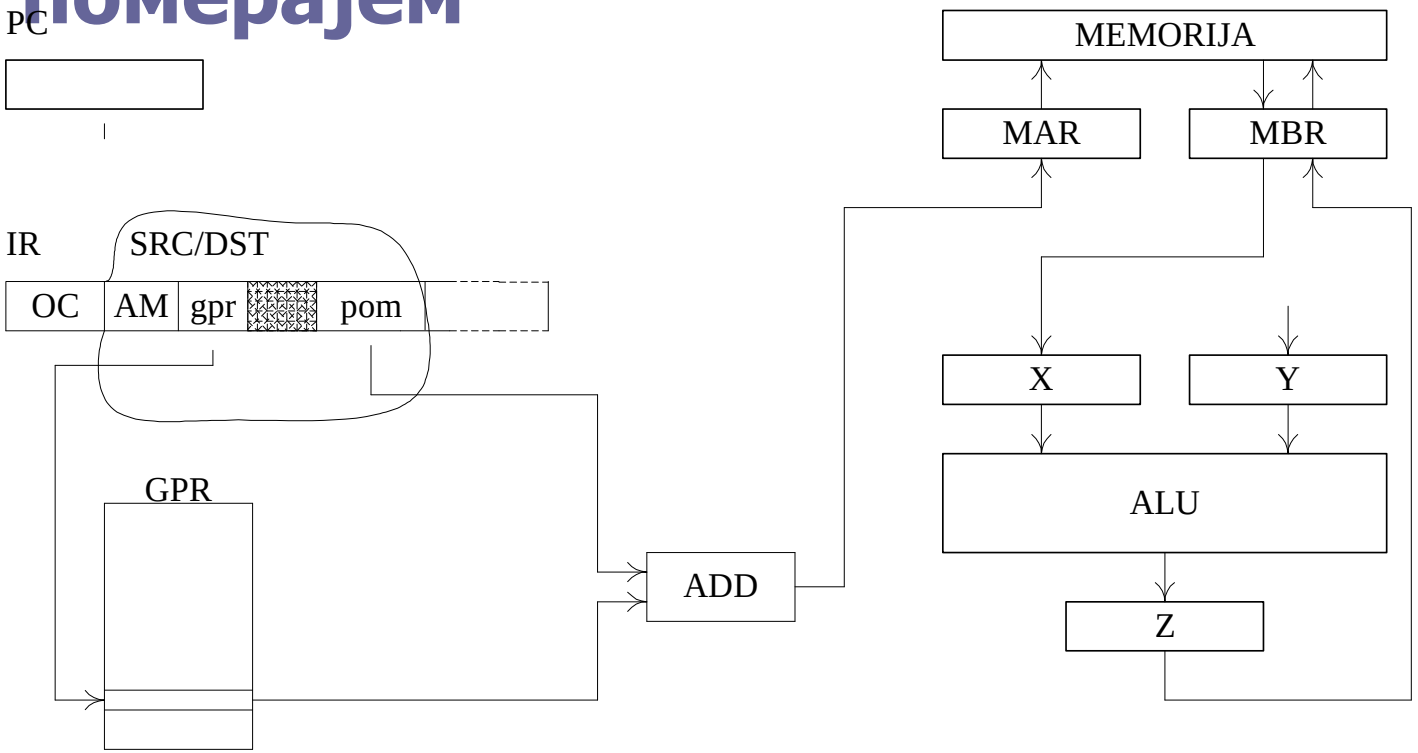
Регистарско индиректно адресирање



Регистарско индиректно са померајем

- Регистарско индиректно адресирање са померајем је адресирање код кога се операнд налази у меморији, а адреса меморијске локације се добија сабирањем садржаја једног од регистара опште намене и помераја.

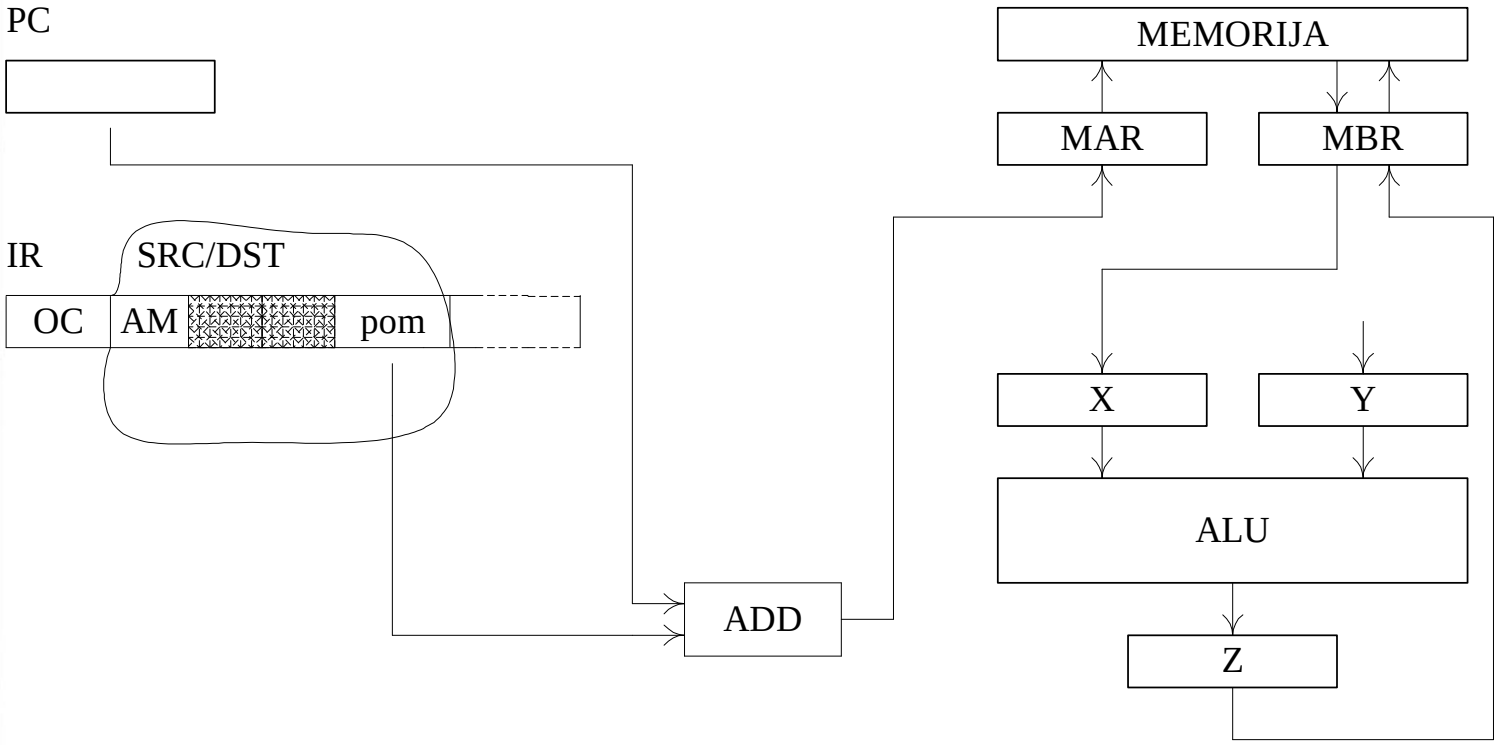
Регистарско индиректно са померајем



Релативно адресирање

- Релативно адресирање са померајем је адресирање код кога се операнд налази у једној од меморијских локација, а адреса меморијске локације се добија сабирањем садржаја програмског бројача PC и помераја.

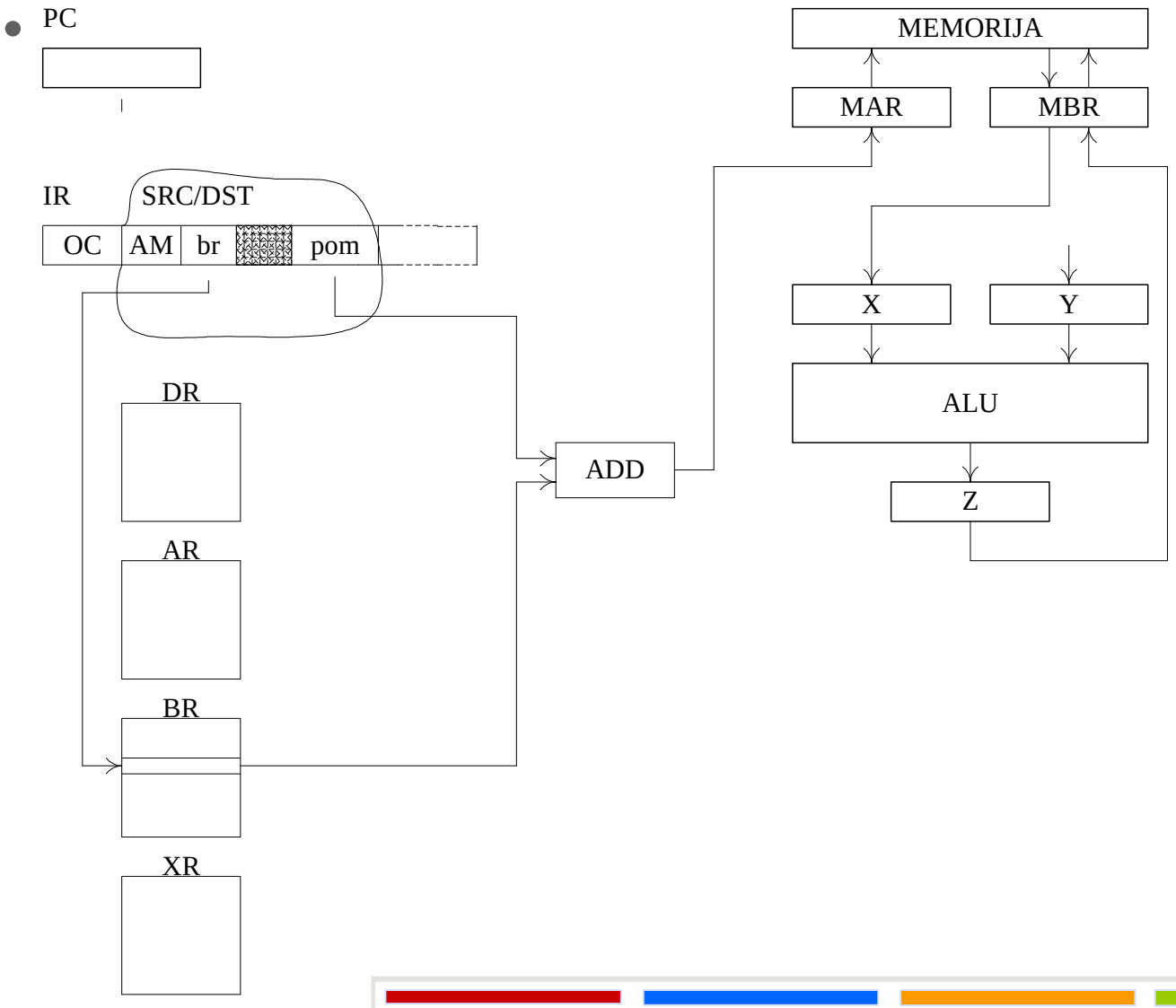
Релативно адресирање са померајем



Базно адресирање са померајем

- Базно адресирање са померајем је адресирање код кога се операнд налази у меморији, а адреса меморијске локације се добија сабирањем садржаја једног од базних регистара и помераја.

Базно адресирање са померајем

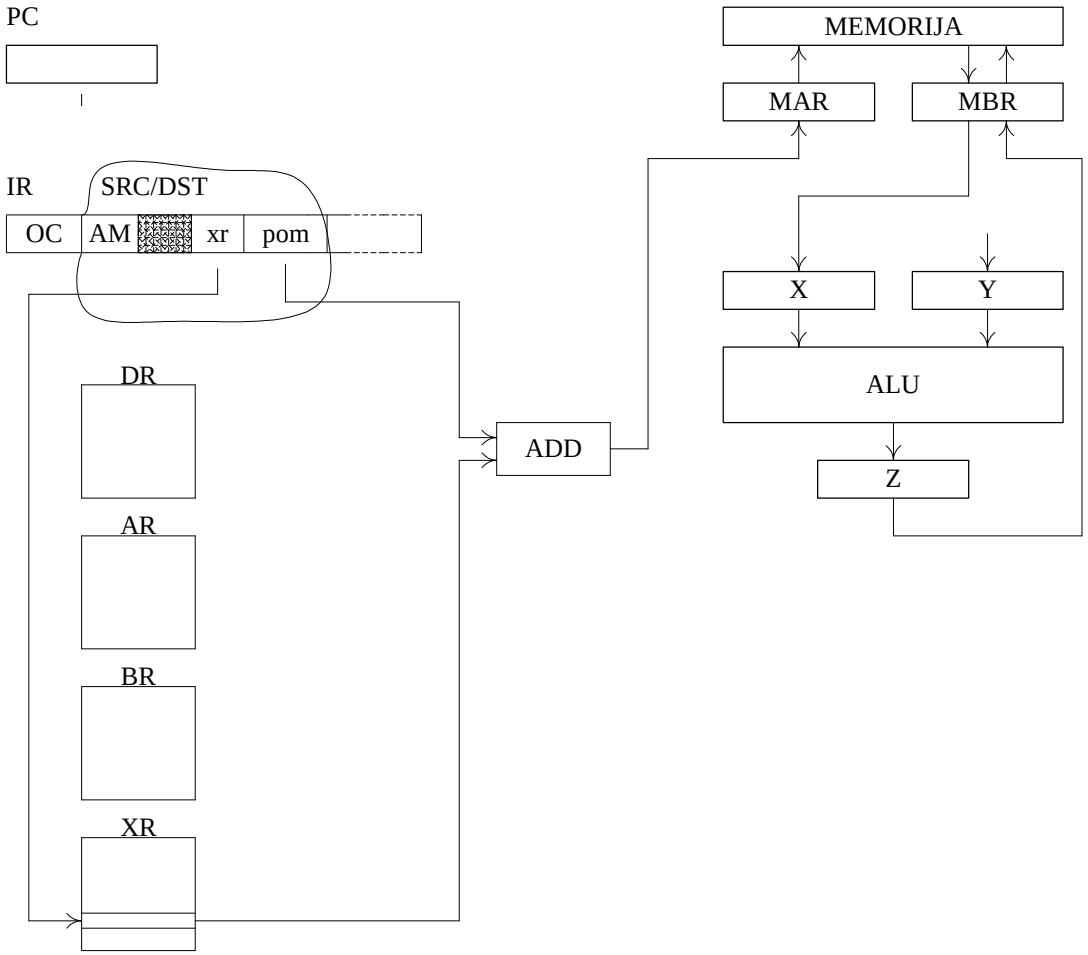


Индексно адресирање са померајем

- Индексно адресирање са померајем је адресирање код кога се операнд налази у меморији, а адреса меморијске локације се добија сабирањем садржаја једног од индексних регистара и помераја.

*Уколико је величина операнда већа од онога што може у једном циклусу да се прочита из меморије индексни регистар може да се помножити величином операнда.
У том случају се адресирање назива скалирано.

Индексно адресирање са померајем

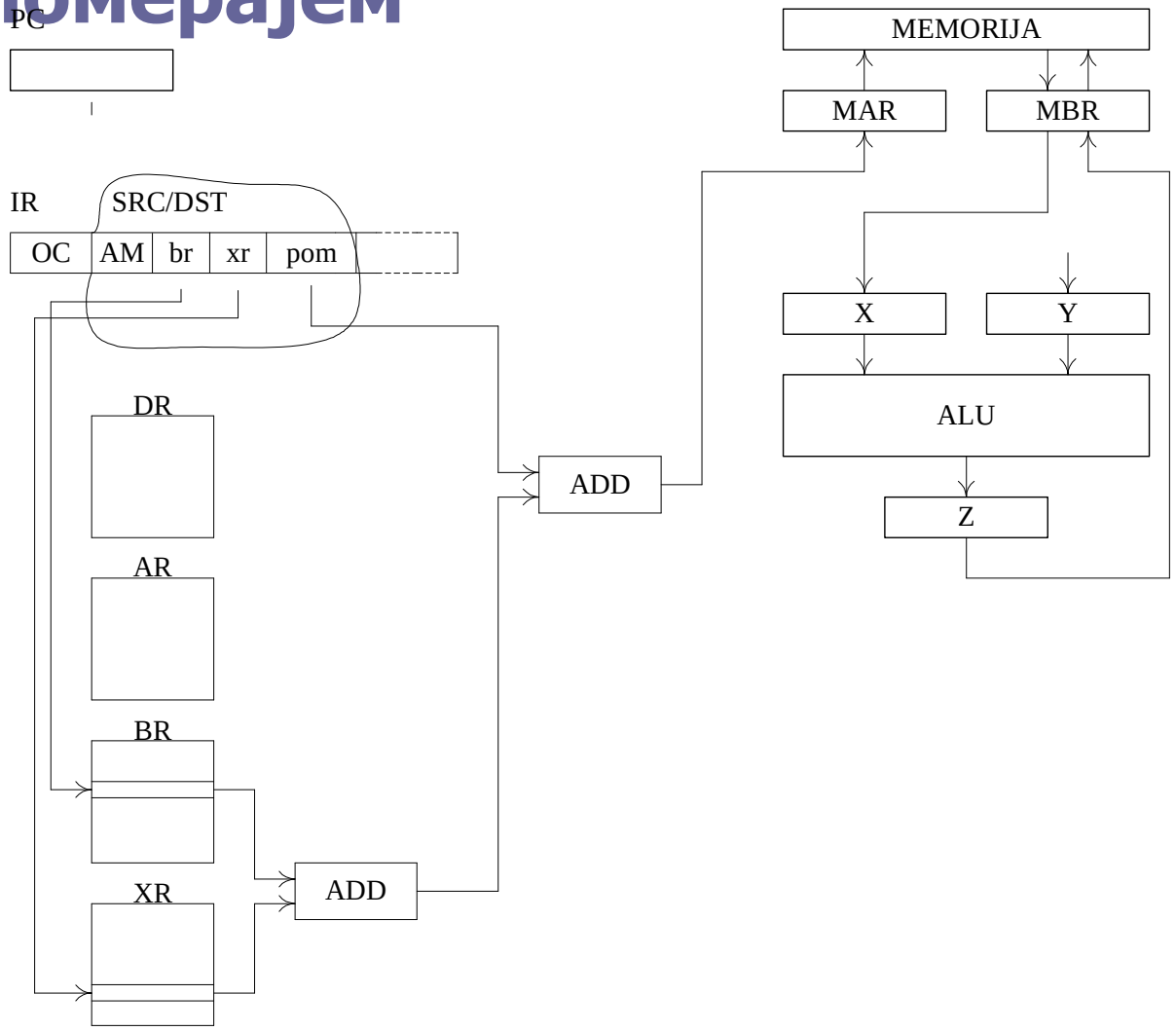


Базно-индексно адресирање са померајем

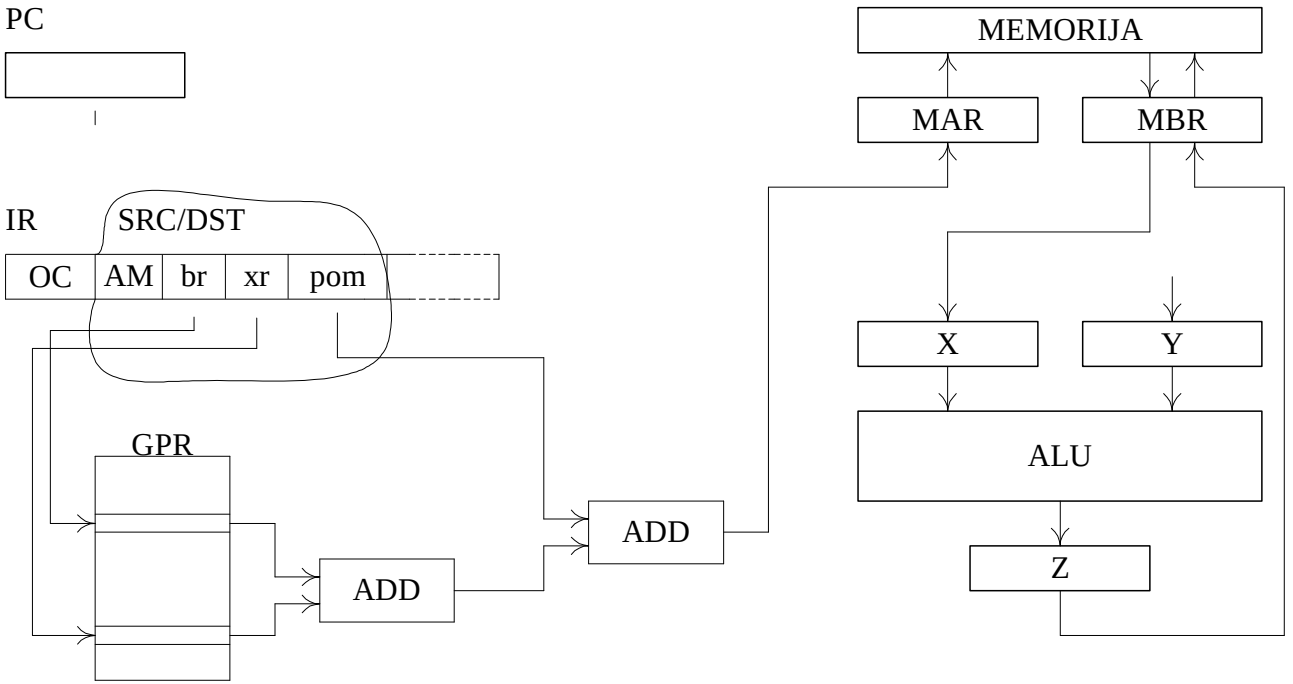
- Базно индексно адресирање са померајем је адресирање код кога се операнд налази у меморији, а адреса меморијске локације се добија сабирањем садржаја једног од базних регистара, једног од индексних регистара и помераја или сабирањем садржаја два регистра опште намене процесора и помераја.

*Уколико је величина операнда већа од онога што може у једном циклусу да се прочита из меморије индексни регистар може да се помножити величином операнда.
У том случају се адресирање назива скалирано.

Базно-индексно адресирање са померајем



Базно-индексно адресирање са померајем



Регистарска индиректна са ауто inc/dec

- Регистарска индиректна адресирања са аутоинкрементирањем и аутодекрементирањем су веома слична регистарском индиректном адресирању, јер се и код ових адресирања као и код регистарског индиректног адресирања операнд налази у једној од меморијских локација, а адреса меморијске локације се налази у једном од регистара.

Регистарска индиректна са ауто inc/dec

- У зависности од тога да ли се прво инкрементира регистар па после тога његов садржај користи као адреса или се прво садржај регистра користи као адреса па се после тога инкрементира, регистарско индиректно адресирање са аутоинкрементирање се јавља као преинкремент адресирање и постинкремент адресирање, респективно.

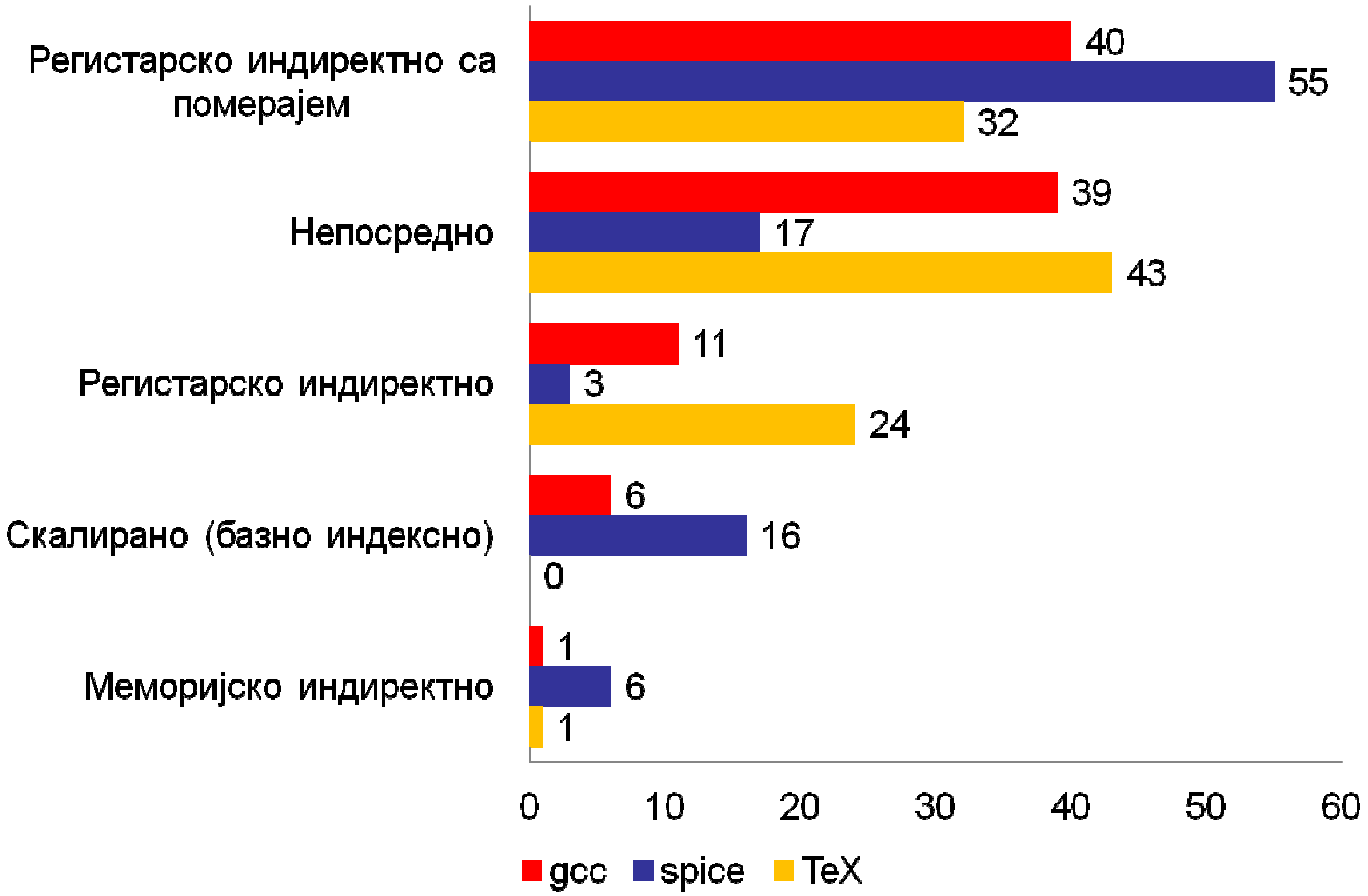
Адресирања

Начин адресирања	Пример	Значење	Типично коришћење
Меморијско директно (memdir)	Load 100h	ACC <= Mem[100h]	Приступа статичким подацима
Меморијско индиректно (memindir)	Load (100h)	ACC <= Mem[Mem[100h]]	Приступа коришћењем показивача
Непосредно (imm)	Load #100h	ACC <= 100h	За константе
Регистарско дректно (regdir)	Load R1	ACC <= R1	Када је податак регистру
Регистарско индиректно (regind)	Load (R1)	ACC <= Mem[R1]	Прступа помоћу показивача који је у регистру
Регистарско индиректно са померајем (reginddisp)	Load (R1)100h	ACC <= Mem[R1+100h]	За приступ локалним променљивама (Симулира меморијско директно (за R1=0) и регистарско индиректно (за померај једнак нули)
Релативно (rel)	Load (PC)100h	ACC <= Mem[PC+100h]	Приступ литералима (константама) и за релативне скокове

Адресирања

Начин адресирања	Пример	Значење	Типично коришћење
Базно са померајем (base)	Load (BP)100h	ACC <= Mem[BP+100h]	Приликом приступа локалним подацима у текућем блоку
Индексно са померајем (index)	Load (XR)100h	ACC <= Mem[XR+100h]	Приликом приступа елементима низа
Базно-индексно (baseindex)	Load (BP,XR)	ACC <= Mem[BP+XR]	Приликом приступа елементима низа који су у локалном блоку
Аутоинкрементирање (posrinc)	Load (R1)+	ACC <= Mem[R1]; R1 <= R1 + d	За обилазан елемената низа који је почев од адресе на коју указује регстар како би се остварило померање на нардни елемент
Аутодекрементирање (predec)	Load -(R1)	R1 <= R1 - d; ACC <= Mem[R1]	За обилазан елемената низа за померањем на претходни елемент. auto inc/dec могу симулирати стек

Адресирања


Регистарско индиректно са померајем
40
55
32
Непосредно
39
17
43
Регистарско индиректно
11
3
24
Скалирано (базно индексно)
6
16
0
Меморијско индиректно
1
6
1
0
10
20
30
40
50
60
gcc
spice
TeX

Распоред бајтова (Byte ordering)

- Уколико се посматра целобројна величина дужине два бајта да ли се у меморију смешта прво млађи или старији бајт?

- Користе се два основна начина за смештање:
  - Уколико млађи бајт смешта на нижу адресу ради се о ***little endian*** поретку.
  - Уколико старији бајт смешта на нижу адресу ради се о ***big endian*** поретку.

Распоред бајтова

- Пример: у меморију почев од адресе 0x100 је потребно сместити целобројну величину дужине 4 бајта 0x12345678

- Уколико млађи бајт иде на нижу адресу:

Адреса	100	101	102	103
Little endian	78	56	34	12

- Уколико старији бајт иде на нижу адресу:

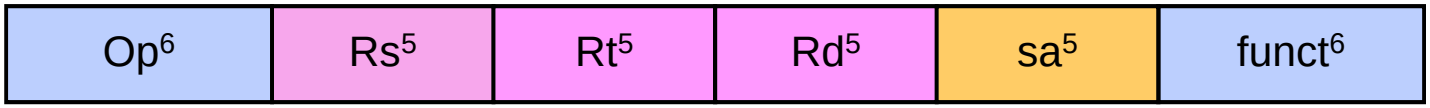
Адреса	100	101	102	103
Big endian	12	34	56	78

Пример MIPS

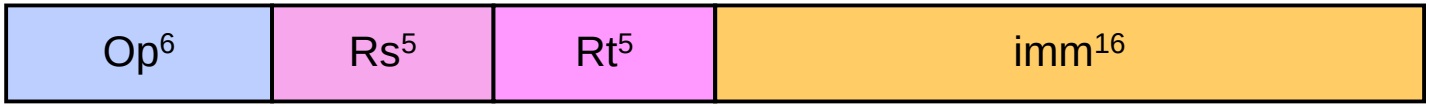
- MIPS
  (Microprocessor without Interlocked Pipeline Stages):
  - MIPS Technologies (некада MIPS Computer Systems, Inc.)
  - Reduced Instruction Set Computer (RISC)
  - Фиксна дужина инструкција
  - Load/Store архитектура
  - Рад са подацима на нивоу регистар-регистар
  - Инструкције користе до 3 регистра
  - Регистара опште намене има 16 (R0=0), дужине 32 бита
  - Подржана оба распореда бајтова: *big endian* + *little endian*
  - Постоји и 64 битни, који се мало разликује
  - ...

MIPS – формат инструкција

- Три формата:
  - Register (R-Type)



  - Immediate (I-Type)



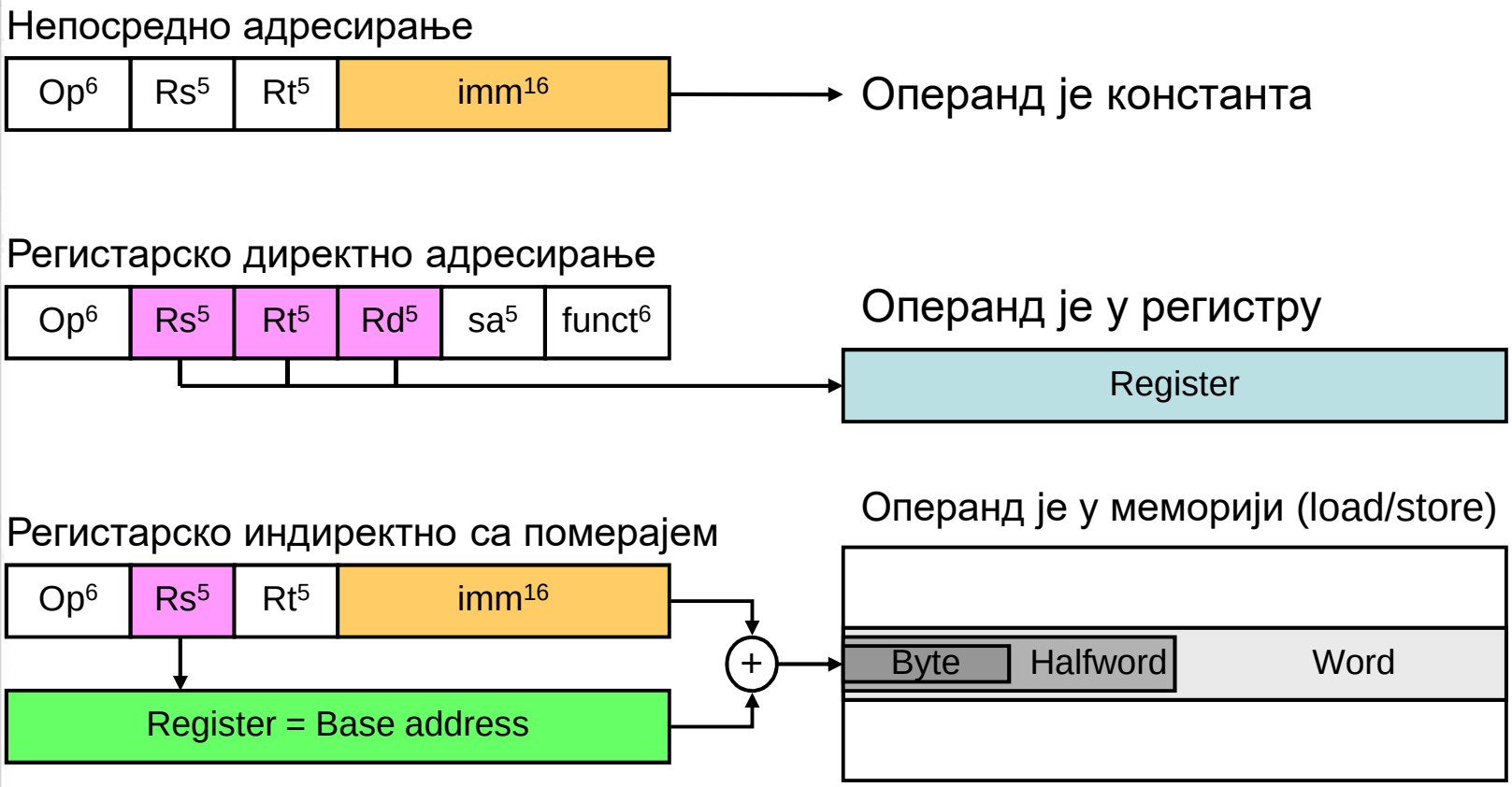
  - Jump (J-Type)

- Све инструкције су дужине 32 бита.

MIPS – формат инструкција

- Op: код операције (opcode)
  - Специфицира операцију коју обавља инструкције
  - Такође специфицира и формат инструкције
- funct: код функције – проширује код операције
  - Проширује $2^6 = 64$ функција за исто поље кода операције
  - MIPS користи opcode 0 за R тип инструкција
- Rs, Rt: први и други изворишни регистар (R тип)
- Rt: одредишни регистар (за I тип)
- Rd: одредишни регистар (R тип)
- sa: за колико треба померити операнд (померачке)
- imm: непосредна величина (кратка константа) или померај код скокова

MIPS – начини адресирања



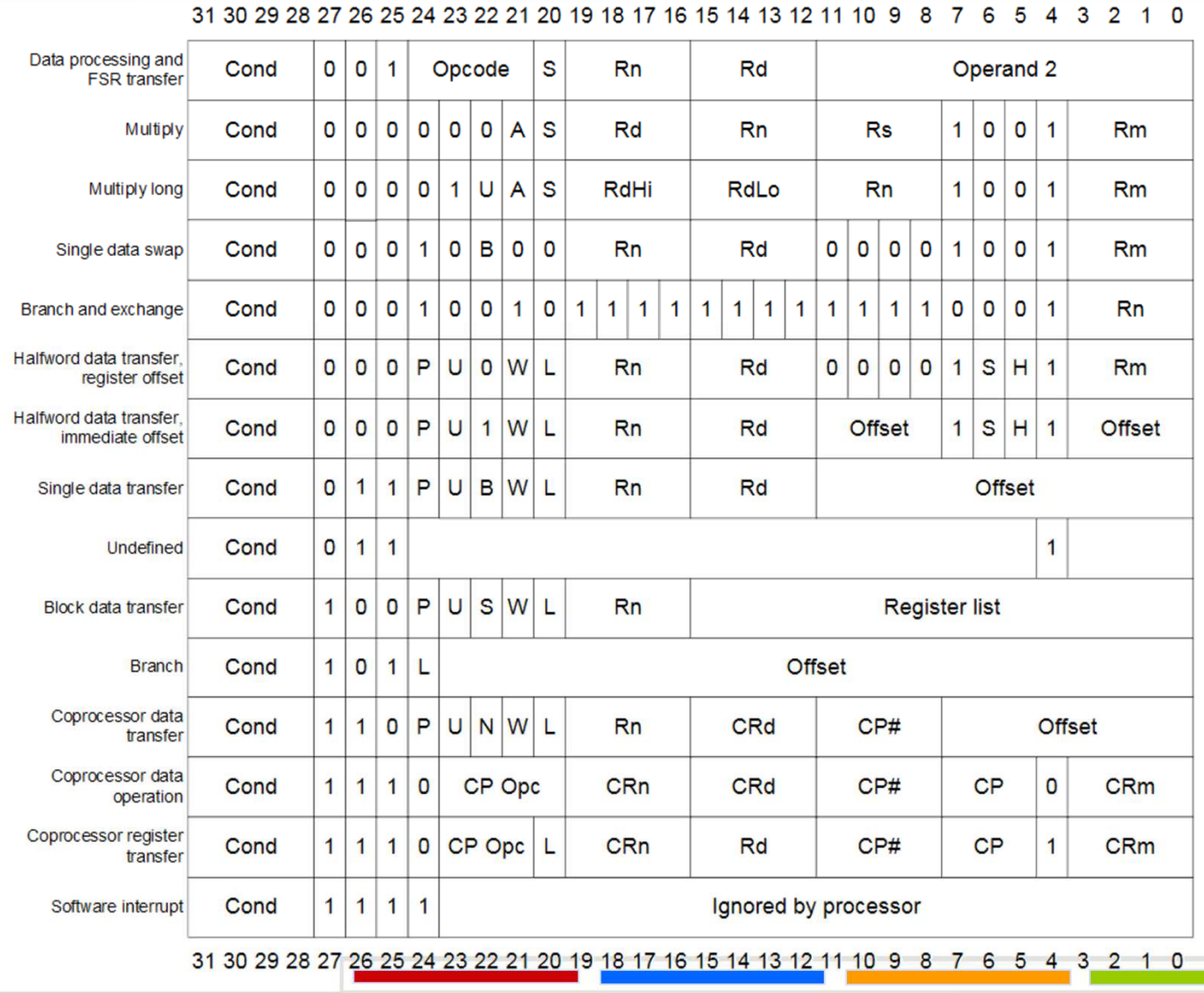
MIPS – скуп инструкција (део)

Категорија	Инструкција	Пример	Значење	Коментар
Arithmetic	add	add $s1, $s2, $s3	$s1 = $s2 + $s3	Three register operands
	subtract	sub $s1, $s2, $s3	$s1 = $s2 - $s3	Three register operands
	add immediate	addi $s1, $s2, 20	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	lw $s1, 20($s2)	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	sw $s1, 20($s2)	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	lh $s1, 20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	lhu $s1, 20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	sh $s1, 20($s2)	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	lb $s1, 20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	lbu $s1, 20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	sb $s1, 20($s2)	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	ll $s1, 20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	sc $s1, 20($s2)	Mernory[$s2+20] = $s1; $s1 = 0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	lui $sl, ,20	$s1 = 20 * 216	Loads constant in upper 16 bits
Logical	and	and $s1, $s2, $s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1, $s2, $s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1, $s2, $s3	$s1 = ~($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1, $s2, 20	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	ori $s1, $s2, 20	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	sll $s1, $s2, 10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1, $s2, 10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1, $s2, 25	if ($ s 1 = = $s2) go to PC + 4 + 100	Equal test: PC-relative branch
	branch on not equal	bne $s1, $s2, 25	if ($s1 ! = $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1, $s2, $s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than: for beq, bne
	set on less than unsigned	sltu $s1, $s2, $s3	if($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	slti $s1, $s2, 20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	sltiu $s1, $s2, 20	if($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$rd = PC + 4; go to 10000	For procedure call

Пример ARM

- AMR:
  - ARM Holdings
  - Reduced Instruction Set Computer (RISC)
  - Фиксна дужина инструкција 32 бита (Thumb* и 16 и 32)
  - Load/Store архитектура
  - Инструкције користе до 3 регистра (некада 4)
  - Рад са подацима углавном на нивоу регистар-регистар
  - Регистара опште намене има 16, дужине 32 бита
  - Подржава: big endian + little endian, подразумевано little
  - Инструкције се условно извршавају (4 битни услов)
  - ...

ARM – формат инструкција



ARM – регистри

- Регистар R13 има улогу регистра SP (Stack Pointer)
- Регистар R14 има улогу да при позивима потпрограма чува повратну адресу LR (Link Register)
- Регистар R15 има улогу програмског бројача PC (Program Counter)
- Програмска статусна реч cpsr (Current Program Status Pegister)
- …

ARM – скуп инструкција (део)

Категорија	Мнемоник	Инструкција	Значење
Arithmetic	ADC	Add with carry	Rd := Rn + Op2 + Carry
	ADD	Add	Rd := Rn + Op2
	MUL	Multiply	Rd := Rm * Rs
	RSB	Reverse Subtract	Rd := Op2 – Rn
	RSC	Reverse Subtract with Carry	Rd := Op2 - Rn - 1 + Carry
	SBC	Subtract with Carry	Rd := Rn - Op2 - 1 + Carry
	CMN	Compare Negative	CPSR flags := Rn + Op2
	CMP	Compare	CPSR flags := Rn - Op2
	MLA	Multiply Accumulate	Rd := (Rm * Rs) + Rn
	SUB	Subtract	Rd := Rn - Op2
Logic	AND	AND	Rd := Rn AND Op2
	EOR	Exclusive OR	Rd := (Rn AND NOT Op2) OR (op2 AND NOT Rn)
	ORR	OR	Rd := Rn OR Op2
	BIC	Bit Clear	Rd := Rn AND NOT Op2
	TEQ	Test bitwise equality	CPSR flags := Rn EOR Op2
	TST	Test bits	CPSR flags := Rn AND Op2
Branches	B	Branch	R15 := address
	BL	Branch with Link	R14 := R15, R15 := address
	BX	Branch and Exchange	R15 := Rn, T bit := Rn[0]
Data transfer	LDR	Load register from memory	Rd := (address)
	STR	Store register to memory	<address> := Rd
	MOV	Move register or constant	Rd : = Op2
	LDM	Load multiple registers	Stack manipulation (Pop)
	STM	Store Multiple	Stack manipulation (Push)
	SWP	Swap register with memory	Rd := [Rn], [Rn] := Rm

ARM-услови извршавања инструкције

Суфикс	Значење	Услов
EQ	Једнако	Z = 1
NE	Није једнако	Z = 0
CS	Веће или једнако (неозначено)	C = 1
CC	Мање (неозначено)	C = 0
MI	Негатив	N = 1
PL	Позитивно или нула	N = 0
VS	Прекорачење	V = 1
VC	Нема прекорачења	V = 0
HI	Веће (неозначено)	(C = 1) $\wedge$ (Z = 0)
LS	Мање или једнако (неозначено)	(C = 0) $\vee$ (Z = 1)
GE	Веће или једнако	N = V
LT	Мање од	N <> V
GT	Веће од	(Z = 0) $\wedge$ (N = V)
LE	Мање или једнако	(Z = 1) $\vee$ (N <> V)
AL	Увек	-

AMR – начини адресирања

- Регистарско директно
- Регистарско индиректно
- Регистарско индиректно са померајем
- Пре/пост инкрементирање/декрементирање
- Скалирано
- PC релативно
- Непосредно адресирање (кратка константа)

Пример x86

- x86:
  - Intel (делимично AMD)
  - Complex Instruction Set Computers (CISCs)
  - Рад са подацима углавном на нивоу регистар-регистар
  - 1978: Intel 8086 16 битна архитектура
  - 1980: 8087 променљиве у покретном зарезу
  - 1982: 80286 адресу постају дужине 24 бита, нове инструкције
  - 1985: 80386 прелазак на дужину 32 бита, нови начини адресирања
  - 1989-1995: 80486, Pentium, Pentium Pro нове инструкције
  - 1997: “MMX” са 57 нових инструкција, Pentium II
  - 1999: Pentium III са нових 70 инструкција
    (SSE – streaming SIMD extensions)
  - 2001: Нових 144 инструкција (SSE2)
  - 2003: AMD адресе постају 64 бита, такође и регистри(AMD64)
  - 2004: Intel прихвата AMD64 (EM64T) и додаје нове инструкције
  - Подржава: little endian

Пример x86

- Комплексност:
  - Инструкције су дужине од 1 до 17 бајтова
  - Један операнд може бити и извориште и одредиште (двоадресни формат)
  - Један операнд може бити из меморије
  - Комплексни начини адресирања (пример: помераји могу бити и 8 и 32 бита)
- Олакшавајуће карактеристике:
  - Најфреквентније инструкције су једноставне
  - Преводиоци у зависности од платформе игноришу поједине делове скупа инструкција

8086 – скуп инструкција (део)

Категорија	Инструкција	Операнди	Значење	Коментар
Aritmetic	ADD	dt, sc	Add	(1) r/m += r/imm; (2) r += m/imm;
	SUB	dt, sc	Subtraction	(1) r/m -= r/imm; (2) r -= m/imm;
	MUL	sc	Unsigned multiply	(1) DX:AX = AX * r/m; (2) AX = AL * r/m;
	DIV	sc	Unsigned divide	DX:AX = DX:AX / r/m; resulting DX = remainder
	CMP	dt, sc	Compare	(1) r/m - r/imm; (2) r - m/imm;
	INC	dt	Increment	r/m ++;
	DEC	dt	Decrement	r/m --;
	NEG	dt	Negate	r/m *= -1;
Logic	AND	dt, sc	Logical AND	(1) r/m &= r/imm; (2) r &= m/imm;
	OR	dt, sc	Logical OR	(1) r/m |= r/imm; (2) r |= m/imm;
	XOR	dt, sc	Logical exclusive OR	(1) r/m ^= r/imm; (2) r ^= m/imm;
	NOT	dt	Logical NOT	r/m ^= -1;
	TEST	dt, sc	Test (logical AND)	(1) r/m & r/imm; (2) r & m/imm;
	SAL	dt, cnt	Shift arithmetic left	(1) r/m <<= 1; (2) r/m <<= CL;
	SHL	dt, cnt	Shift logical left	
	SAR	dt, cnt	Shift arithmetic right	(1) (signed)r/m >>= 1; (2) (signed)r/m >>= CL;
	SHR	dt, cnt	Shift logical right	
	ROL	dt, cnt	Rotate left	
	ROR	dt, cnt	Rotate right	
Data transfer	MOV	dt, sc	Move	(1) r/m = r; (2) r = r/m;
	IN	ac, port	Input from port	(1) AL = port[imm]; (2) AL = port[DX]; (3) AX = port[DX];
	OUT	port, sc	Output to port	(1) port[imm] = AL; (2) port[DX] = AL; (3) port[DX] = AX;
	POP	dt	Pop word off stack	r/m = *SP++;
	PUSH	sc	Push word onto stack	*--SP = r/m;
Jump	JMP	label	Jump	
	Jcc	slabel	Jump if condition	(JA, JAE, JB, JBE, JC, JE, JG, JGE, JL, JLE, JNA, JNAE, JNB, JNBE, JNC, JNE, JNG, JNGE, JNL, JNLE, JNO, JNP, JNS, JNZ, JO, JP, JPE, JPO, JS, JZ)
	CALL	proc	Call a procedure	push eip;
	RET	[pop]	Return from procedure	
	IRET		Return from interrupt	

8086 – скуп инструкција (део)

OPCODE		DESCRIPTION
AAA		ASCII adjust addition
AAD		ASCII adjust division
AAM		ASCII adjust multiply
AAS		ASCII adjust subtraction
ADC	dt, sc	Add with carry
ADD	dt, sc	Add
AND	dt, sc	Logical AND
CALL	proc	Call a procedure
CBW		Convert byte to word
CLC		Clear carry flag
CLD		Clear direction flag
CLI		Clear interrupt flag
CMC		Complement carry flag
CMP	dt, sc	Compare
CMPS	[dt, sc]	Compare string
CMPSB	[dt, sc]	Compare string bytes
CMPSW	[dt, sc]	Compare string words
CWD		Convert word to double word
DAA		Decimal adjust addition
DAS		Decimal adjust subtraction
DEC	dt	Decrement
DIV	sc	Unsigned divide
ESC	code, sc	Escape
HLT		Halt
IDIV	sc	Integer divide
IMUL	sc	Integer multiply
IN	ac, port	Input from port
INC	dt	Increment
INT	type	Interrupt
INTO		Interrupt if overflow
IRET		Return from interrupt

OPCODE		DESCRIPTION
JA	slabel	Jump if above
JAE	slabel	Jump if above or equal
JB	slabel	Jump if below
JBE	slabel	Jump if below or equal
JC	slabel	Jump if carry
JCXZ	slabel	Jump if CX is zero
JE	slabel	Jump if equal
JG	slabel	Jump if greater
JGE	slabel	Jump if greater or equal
JL	slabel	Jump if less
JLE	slabel	Jump if less or equal
JMP	label	Jump
JNA	slabel	Jump if not above
JNAE	slabel	Jump if not above or equal
JNB	slabel	Jump if not below
JNBE	slabel	Jump if below or equal
JNC	slabel	Jump if no carry
JNE	slabel	Jump if not equal
JNG	slabel	Jump if not greater
JNGE	slabel	Jump if not greater or equal
JNL	slabel	Jump if not less
JNLE	slabel	Jump if not less or equal
JNZ	slabel	Jump if not zero
JNO	slabel	Jump if not overflow
JNP	slabel	Jump if not parity
JNS	slabel	Jump if not sign
JO	slabel	Jump if overflow
JPO	slabel	Jump if parity odd
JP	slabel	Jump if parity
JPE	slabel	Jump if parity even
JS	slabel	Jump if sign
JZ	slabel	Jump if zero

OPCODE		DESCRIPTION
LAHF		Load AH from flags
LDS	dt, sc	Load pointer using DS
LEA	dt, sc	Load effective address
LES	dt, sc	Load pointer using ES
LOCK		Lock bus
LODS	[se]	Load string
LODSB	[se]	Load string bytes
LODSW	[se]	Load string words
LOOP	slabel	Loop
LOOPE	slabel	Loop if equal
LOOPZ	slabel	Loop if zero
LOOPNE	slabel	Loop if not equal
LOOPNZ	slabel	Loop if not zero
MOV	dt, sc	Move
MOVS	[dt, sc]	Move string
MOVSB	[dt, sc]	Move string bytes
MOVSW	[dt, sc]	Move string words
MUL	sc	Unsigned multiply
NEG	dt	Negate
NOP		No operation
NOT	dt	Logical NOT
OR	dt, sc	Logical OR
OUT	port, sc	Output to port
POP	dt	Pop word off stack
POPF		Pop flags off stack
PUSH	sc	Push word onto stack
PUSHF		Push flags onto stack
RCL	dt, cnt	Rotate left through carry
RCR	dt, cnt	Rotate right through carry
REP		Repeat string operation
REPE		Repeat while equal
REPZ		Repeat while zero

OPCODE		DESCRIPTION
REPNE		Repeat while not equa
REPNZ		Repeat while not zero
RET	[pop]	Return from procedure
ROL	dt, cnt	Rotate left
ROR	dt, cnt	Rotate right
SAHF		Store AH into flags
SAL	dt, cnt	Shift arithmetic left
SHL	dt, cnt	Shift logical left
SAR	dt, cnt	Shift arithmetic right
SBB	dt, sc	Subtract with borrow
SCAS	[dt]	Scan string
SCASB	[dt]	Scan string byte
SCASW	[dt]	Scan string word
SHR	dt, cnt	Shift logical right
STC		Set carry flag
STD		Set direction flag
STI		Set interrupt flag
STOS	[dt]	Store string
STOSB	[dt]	Store string byte
STOSW	[dt]	Store string word
SUB	dt, sc	Subtraction
TEST	dt, sc	Test (logical AND)
WAIT		Wait for 8087
XCHG	dt, sc	Exchange
XLAT	table	Translate
XLATB	table	Translate
XOR	dt, sc	Logical exclusive OR

Notes:

dt - destination

sc - source

label - may be near or far address

slabel - near address

x86 - регистри

- AL/AH/AX/EAX/RAX: Акумулатор
- BL/BH/BX/EBX/RBX: Индексни регистар
- CL/CH/CX/ECX/RCX: Бројачки регистар (петње и стрингови)
- DL/DH/DX/EDX/RDX: Проширење прецизности акумулатора
- SI/ESI/RSI: Изворишни индекс за стринг операције
- DI/EDI/RDI: Одредишни индекс за стринг операције
- SP/ESP/RSP: Показивач на врх стека
- BP/EBP/RBP: Базни регистар
- IP/EIP/RIP: Instruction pointer

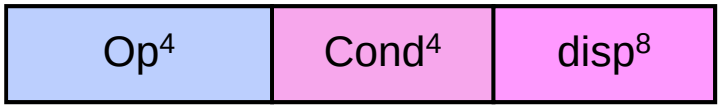
x86 – начини адресирања

- Регистарско директно адресирање
- Непосредно
- Меморијско директно адресирање
- Регистарско индиректно адресирање
- Индесно (скалирано) адресирање
- Базно-индексно (скалирано) адресирање

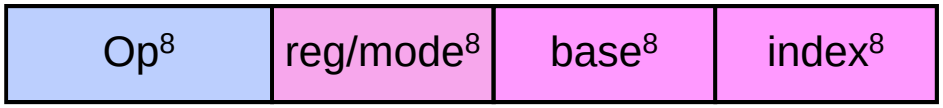
Пример IA-32 (x86)

- PUSH 5-bit opcode, 3-bit register operand

- JE (PC)disp 4-bit opcode, 4-bit condition, 8-bit jump offset



- XOR 8-bit opcode, 8-bit reg/mode*, 8-bit base, 8-b index



Пример Java bytecode

- Јава:
  - Oracle (SUN)
  - Виртуелна машина (негде и хардверска подршка)
  - Стек машина
    (Нула-адресни формат инструкција)
  - Подржава: big endian
  - ...

Java bytecode – скуп инструкција (део)

Категорија	Инструкција	Аргументи	Стек	Коментар
Aritmetic	iadd		value1, value2 → result	add two ints
	isub		value1, value2 → result	int subtract
	imul		value1, value2 → result	multiply two integers
	idiv		value1, value2 → result	divide two integers
	irem		value1, value2 → result	logical int remainder
	ineg		value → result	negate int
Logic	iand		value1, value2 → result	perform a bitwise and on two integers
	ior		value1, value2 → result	bitwise int or
	ixor		value1, value2 → result	int xor
	ishl		value1, value2 → result	int shift left
	ishr		value1, value2 → result	int arithmetic shift right
	iushr		value1, value2 → result	int logical shift right
Data transfer	bipush	1: byte	→ value	push a byte onto the stack as an integer value
	pop		value →	discard the top value on the stack
	dup		value → value, value	duplicate the value on top of the stack
	swap		value2, value1 → value1, value2	swaps two top words on the stack
	iconst_x		→ x	load the int value x onto the stack (-1 <=x<=5)
	iload	1: index	→ value	load an int value from a local variable #index
	iload_x		→ value	load an int value from local variable x (0<=x<=3)
	istore	1: index	value →	store int value into variable #index
	istore_x		value →	store int value into variable x (0<=x<=3)
Jump	goto	2: branchbyte1, branchbyte2	[no change]	goes to another instruction at branchoffset (branchbyte1 << 8 + branchbyte2)
	jsr	2: branchbyte1, branchbyte2	→ address	jump to subroutine at branchoffset and place the return address on the stack
	ireturn		value → [empty]	return an integer from a method
	return		→ [empty]	return void from method
	ifxx	2: branchbyte1, branchbyte2	value →	Conditional branch to branchoffset (eq, ne, lt, ge, gt, le, ...)

Питања?

Електротехнички Факултет
Универзитет у Београду