5b
Aritmetični ukazi

BRANKO ŠTER

- Osnovni nabor ukazov RV32I

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm | pred | succ		rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

Registri RISC-V

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5–7	t0–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

ABI

- Application binary interface
  - vmesnik med program v strojni kodi (tudi programi iz knjižnic in programi OS)
  - Določa način zapisa podatkovnih struktur in način klicanja podprogramov na nizkem nivoju
    - API določa podobne stvari v izvorni kodi
  - ABI določa:
    - strukturo registrov, organizacijo sklada, vrste pomnilniških dostopov
    - podatkovni tipi (velikost, poravnanost, endian)
    - način klica sistemskih klicev
    - dogovor o klicih podprogramov (Calling convention)

ALE ukazi

1. **aritmetične operacije** (+, –)
   - ADD, ADDI,
   - SUB
   - LUI, AUIPC
2. **logične bitne operacije** (&, V, ∇)
   - AND, ANDI
   - OR, ORI
   - XOR, XORI
3. **pomiki** (shift) (levi, desni; logični, aritmetični)
   - SLL, SLLI
   - SRL, SRLI
   - SRA, SRAI
4. **ukazi za primerjavo oz. set operacije** (pogoj: <)
   - SLT, SLTI, SLTU, SLTIU

Logične bitne operacije

- Logične bitne operacije delujejo po istoležnih bitih (bitwise operations):
  - IN (AND), &

```
  00110010
& 01010110
----------
  00010010
```

  - ALI (OR), V, |

```
  00110010
| 01010110
----------
  01110110
```

  - Ekskluzivni ALI (XOR), ∇, ^

```
  00110010
^ 01101001
----------
  01011011
```

  - NE (NOT) – tega RISCV sicer nima, ker se da to narediti z XOR z enicami

```
~00011011
----------
 11100100
```

Uporaba bitnih operacij

Bitne operacije se uporabljajo tudi za branje in vpisovanje posameznih bitov v besedo

- Nastavljanje bita:
  - Kako nastavimo nek bit na 1 (ostale pa pustimo pri miru):

```
    xxxxxxxx
or  00010000
    xxx1xxxx
```

- Brisanje bita:
  - Kako postavimo nek bit na 0 (ostale pa pustimo pri miru):

```
    xxxxxxxx
and 11101111
    xxx0xxxx
```

- Branje bita:
  - Kako samo pogledamo vrednost določenega bita:

```
    xxxxxxxx
and 00010000
    000x0000
```

Če je iskani bit 1, je dobljeni izraz od 0 različen, sicer je 0.

Pomiki

- Pomiki:
  - levi
    - SLL - Shift Left (Logical) in SLLI (SLL immediate)
      - 0110 $\rightarrow$ 1100
  - desni
    - **logični** (0110 $\rightarrow$ 0011)
      - v izpraznjena mesta gredo ničle
    - **aritmetični** (0110 $\rightarrow$ 0011, 1011 $\rightarrow$ 1101)
      - najbolj levi bit se ne spreminja in se vstavlja v izpraznjena mesta (število smatramo kot predznačeno – ta bit je predznak)

- Levi pomik (za n mest) predstavlja tudi množenje z 2^n
  - 00000101 << 3 = 00101000
- Desni pomik (za n mest) pa je deljenje z 2^n
  - 00110010 >> 4 = 00000011
- Aritmetični pomik ohrani predznak
  - število obravnava kot predznačeno
    - 11000 >> 1 = 11100
  - ni pa to več pravo celoštevilsko deljenje!
    - 11001 >> 1 = 11100 (-7 >> 1 = -4)
- S pomiki in seštevanjem/odštevanjem je možno realizirati tudi poljubno množenje/deljenje
- Tudi pomiki (logični) se uporabljajo za izločanje/vstavljanje bitov
  - npr. 0x1 << 2 = 0100

Seznam vseh ALE ukazov

- ALE ukazi so 3-operandni
- 2 operanda sta v registrih
  - tretji je lahko v registru ali takojšnji (immediate)

rd ← rs1 op rs2
dd ← rs1 op Takojšnji operand (immediate)

ALE ukazi (1): aritmetične in logične operacije

Tip operacije	Ukaz	Opis	Format	Polja	Opkoda
Aritmetične	ADD	Add	R	0000000 rs2 rs1 000 rd	0110011
	SUB	Subtract	R	0100000 rs2 rs1 000 rd	0110011
	ADDI	Add imm.	I	imm12 rs1 000 rd	0010011
	LUI	Load upper imm.	U	imm20 rd	0110111
	AUIPC	Add upper imm. PC	U	imm20 rd	0010111
Tip operacije	**Ukaz**	**Opis**	**Format**	**funct7, funct3**	**opkoda**
Logične	AND	And	R	0000000 rs2 rs1 111 rd	0110011
	OR	Or	R	0000000 rs2 rs1 110 rd	0110011
	XOR	Exclusive or	R	0000000 rs2 rs1 100 rd	0110011
	ANDI	And imm.	I	imm12 rs1 111 rd	0010011
	ORI	Or imm.	I	imm12 rs1 110 rd	0010011
	XORI	Excl.-or imm.	I	imm12 rs1 100 rd	0010011

ALE ukazi (2): Pomiki

Tip opera-cije	Ukaz	Opis	Format	Polja	Opkoda
shift	SLL	Shift left logical	R	0000000 rs2 rs1 001 rd	0110011
	SRL	Shift right logical	R	0000000 rs2 rs1 101 rd	0110011
	SRA	Shift right arithmetic	R	0100000 rs2 rs1 101 rd	0110011
	SLLI	Shift left logical imm.	I	0000000 shamt* rs1 001 rd	0010011
	SRLI	Shift right logical immediate	I	0000000 shamt* rs1 101 rd	0110011
	SRAI	Shift right arithmetic imm.	I	0000000 shamt* rs1 101 rd	0110011

*shamt ... shift amount

Ukazi za pomike uporabljajo pomikalnik (barrel shifter)

- kombinacijsko vezje, ki izvede poljuben pomik (za 0, ..., 31 mest) v eni urini periodi
- število mest pomika je podano v *rs2* ali v takojšnjem operandu

ALE ukazi (3): Ukazi za primerjavo

Tip operacije	Ukaz	Opis	Format	Polja	Opkoda
set	SLT	Set if less than	R	0000000 rs2 rs1 010 rd	0110011
	SLTU	Set if less than unsigned	R	0000000 rs2 rs1 011 rd	0110011
	SLTI	Set if less than immediate	I	imm12 rs1 010 rd	0010011
	SLTUI	Set if less than unsig. imm.	I	imm12 rs1 011 rd	0010011

Če je pogoj izpolnjen, se v *rd* zapiše 1, sicer 0

ADD:

```
add  x3, x5, x6          ; x3 ← x5 + x6
```

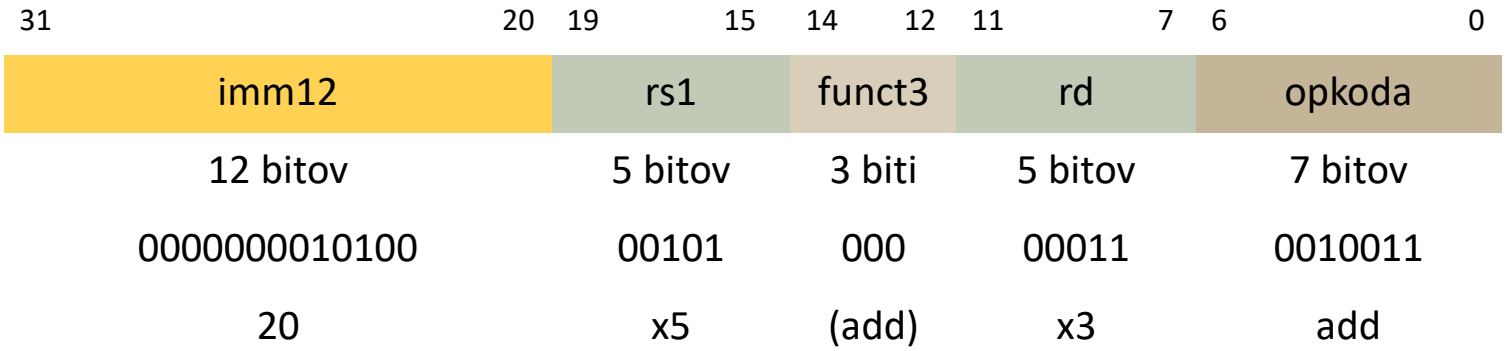
Format R:

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opkoda
7 bitov	5 bitov	5 bitov	3 biti	5 bitov	7 bitov
0000000	00110	00101	000	00011	0110011
(add)	x6	x5	(add)	x3	add

ADDI (Add immediate)

addi x3, x5, 20 ; x3 ← x5 + 20

Format I:

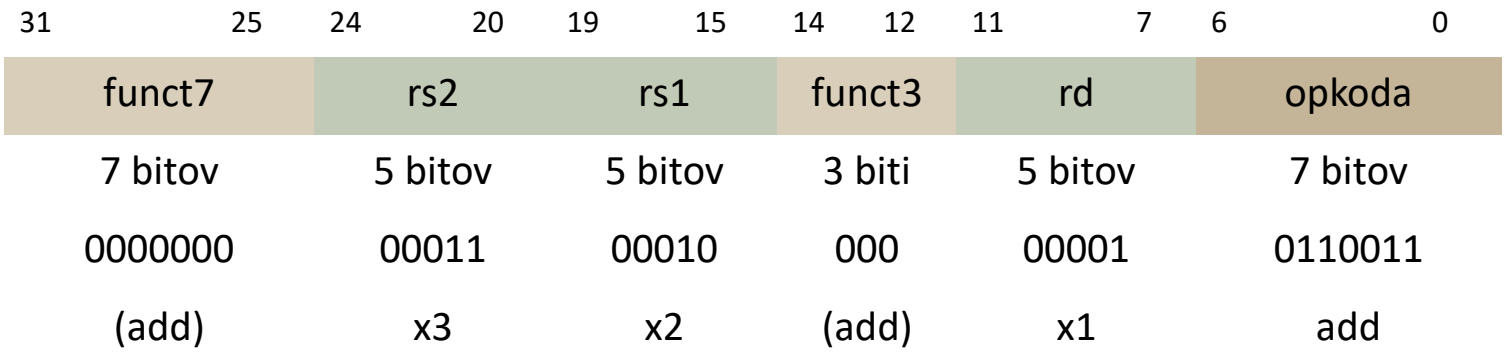


- Takojšnjemu (12-bitnemu) operandu se razširi predznak (na 32 bitov).
- A pozor: če pišemo v šestnajstiškem zapisu, ne sme imeti na začetku enice (zbirnik javi napako)!
  - Če želimo, da je negativen, moramo dodati predznak (npr. -0x800)
  - V desetiškem zapisu ima negativno število itak predznak (npr. imm. -1 se razširi na same enice v registru)

AND

and x1, x2, x3 ; x1 ← x2 & x3

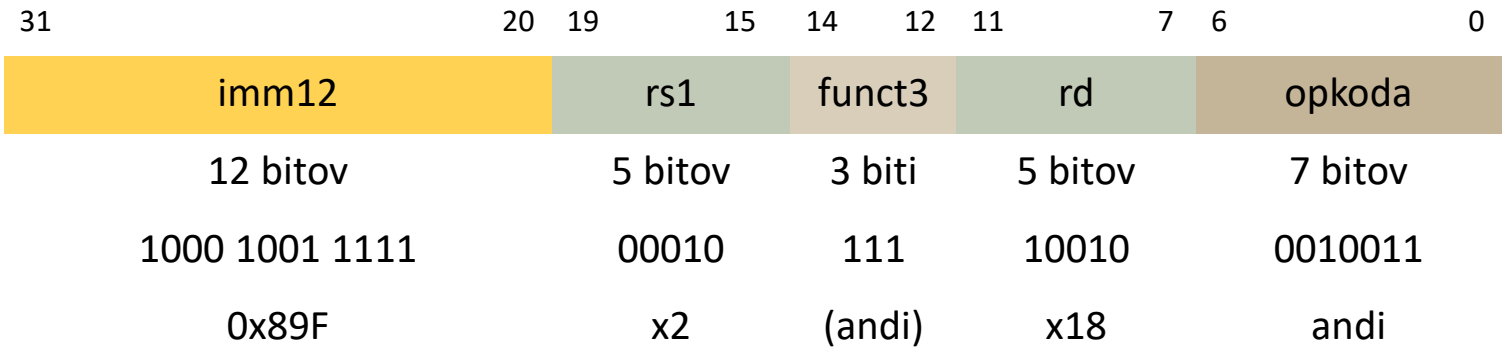
Format R:



ANDI (and immediate)

andi x18, x2, 0x49F ; x18 ← x2 & 0x49F

Format I:

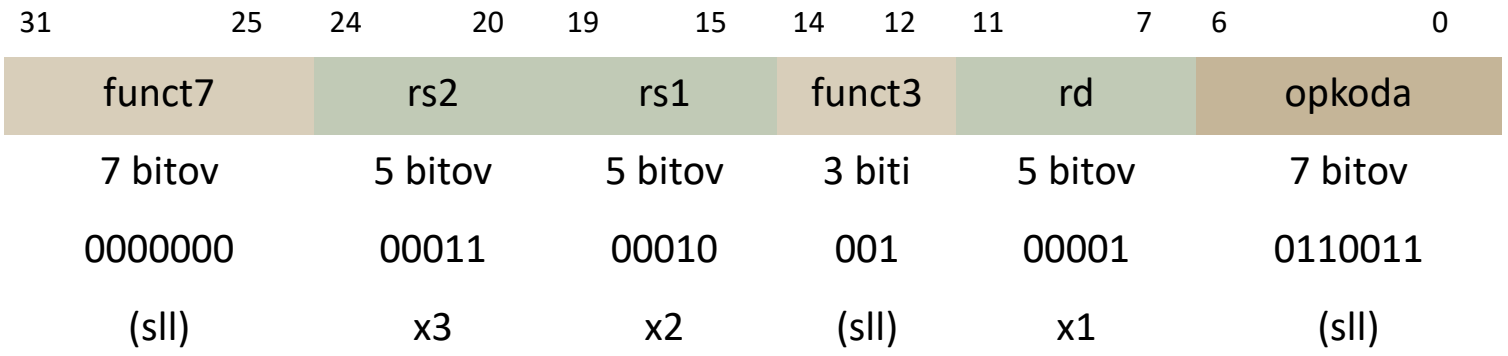


- Takojšnjemu (12-bitnemu) operandu se razširi predznak (na 32 bitov).
- A pozor: če pišemo v šestnajstiškem zapisu, ne sme imeti na začetku enice (zbirnik javi napako)!
  - Če želimo, da je negativen, moramo dodati predznak

SLL (shift left logical)

sll x1, x2, x3 ; $x1 \leftarrow x2 << x3$ (oz. $r2 \times 2^{r3}$)

Format R:

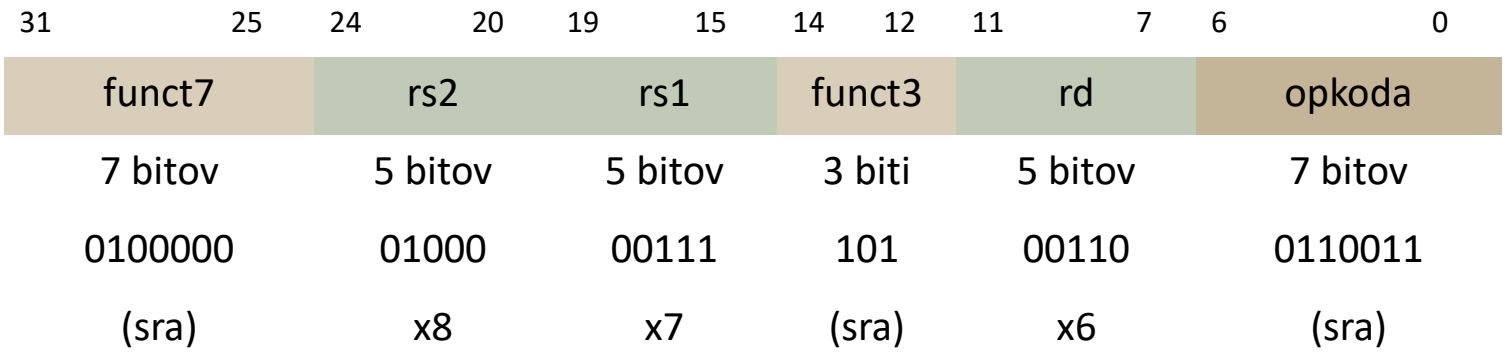


SRA (shift right arithmetic)

sra x6, x7, x8 ; x6 ← x7 >> x8

; $x6_{31} \leftarrow x7_{31}$

Format R:



LUI (Load upper immediate)

- poseben ukaz, ki 20-bitno (konstantno) vrednost naloži v gornjih 20 bitov registra, spodnjih 12 bitov pa je 0
- Zakaj sploh potrebujemo tak ukaz?
  - Problem je, kako naložiti 32-bitno konstanto v register
  - z enim 32-bitnim ukazom ni možno
  - zato to lahko storimo v 2 korakih:
    1. naložimo zgornjih 20 bitov
    2. naložimo spodnjih 12 bitov

  - Npr.: 0x12345678

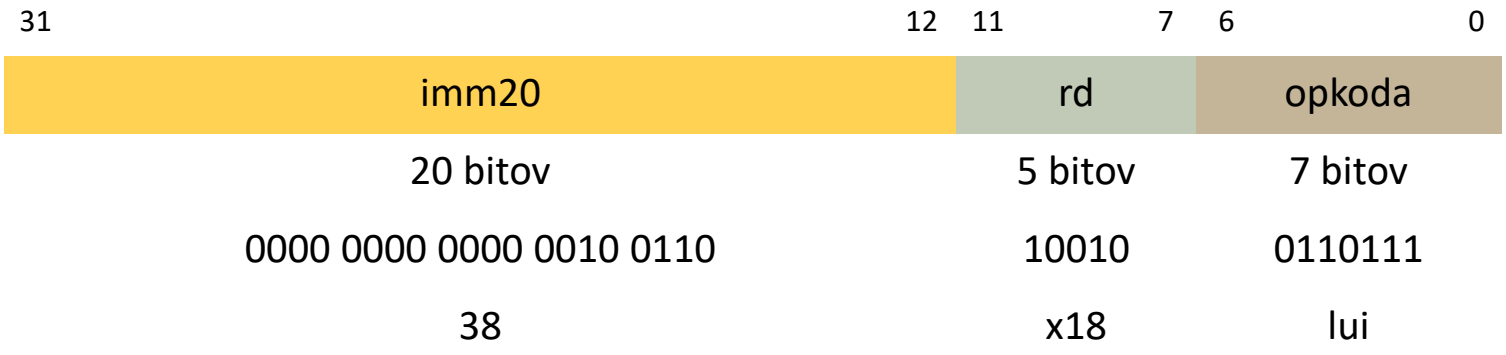
```
lui   x5, 0x12345
addui   x5, x5, 0x678
```

(lahko tudi z `ori`)

LUI (load upper immediate)

lui x18, 38 # $x18_{31..12} \leftarrow 38,\ x18_{11..0} \leftarrow 0$

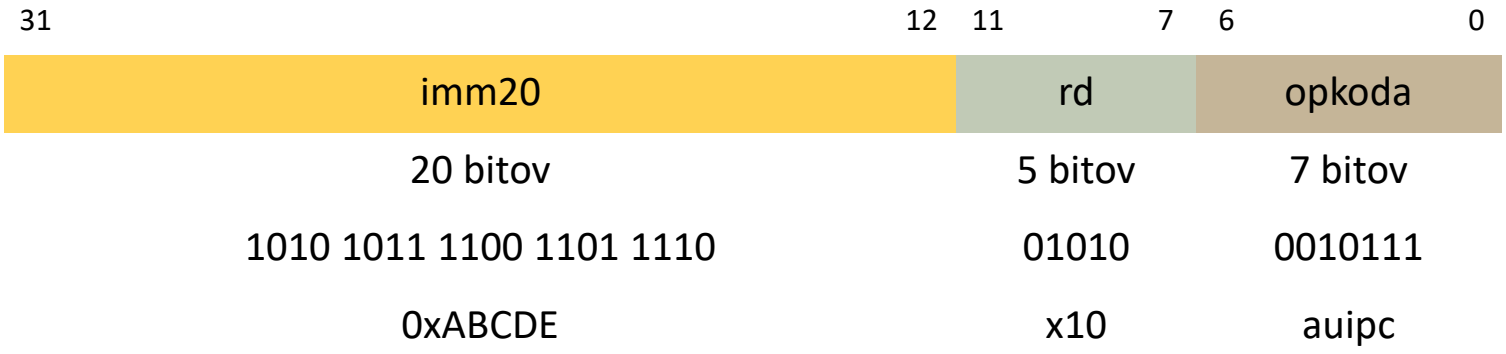
Format U:



AUIPC (add upper immediate to PC)

```
auipc x10, 0xABCDE    # x10 ← (0xABCDE << 12) + PC
```

Format U:



- Tudi to je poseben ukaz, ki 20-bitno (konstantno) vrednost naloži v gornjih 20 bitov registra (spodnjih 12 bitov je 0), temu pa prišteje vrednost programskega števca PC
- Ta ukaz omogoča PC-relativno naslavljanje
  - Tako se da celoten program linearno premakniti v drug del pomnilnika

- Primer: program, ki na osnovi pomikov in seštevanja 32-bitno nepredznačeno spremenljivko A množi z 10 in jo shrani v B:

```
        .data           # (na naslovu 0x400)
A:      .word 5
B:      .word 0

        .text
        lw x1, A(x0)
        slli x2, x1, 3
        slli x3, x1, 1
        add x4, x2, x3
        sw x4, B(x5)  # x5 je začasni register
```

Set-ukazi (oz. ukazi za primerjavo)

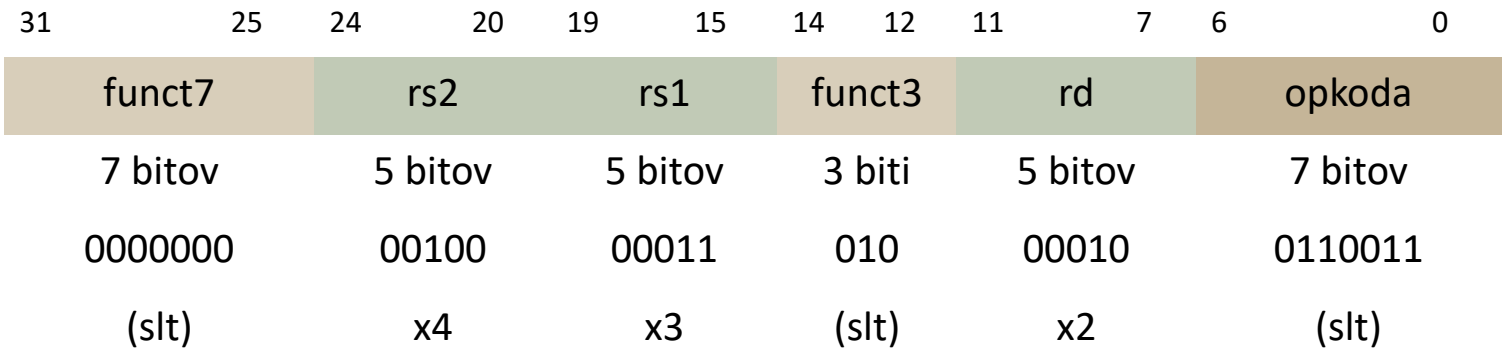
Če je podani pogoj izpolnjen, postavijo v ciljni register 1 (...0001), sicer 0 (...0000)

- SLT (Set if Less Than), format R
  - slt rd, rs1, rs2 ; rd ← (rs1 < rs2) ? 1 : 0
- SLTI (Set if Less Than Immediate), format I
  - slti rd, rs1, imm ; rd ← (rs1 < imm i) ? 1 : 0
- SLTIU (Set if Less Than Immediate Unsigned), format I
  - sltiu rd, rs1, imm ; rd ← (rs1 < imm i) ? 1 : 0
- SLTU (Set if Less Than Unsigned), format R
  - sltu rd, rs1, rs2 ; rd ← (rs1 < rs2) ? 1 : 0

SLT (set if less than)

slt x2, x3, x4 ; x2 ← (x3 < x4)

Format R:



Psevdo-ukazi

- Poleg direktiv obstajajo tudi psevdo-ukazi, ki niso dejanski ukazi procesorja, ampak so namenjeni zbirniku, ki jih prevede v dejanske ukaze
- Primeri:

nop	(addi x0, x0, 0)	no operation
la rd, sym	(auipc+addi)	nalaganje naslova (load address)
li rd, imm		nalaganje konstante (load imm.)
mv rd, rs	(addi rd, rs, 0)	vsebina rs se kopira v rd
not rd, rs	(xori rd, rs, -1)	eniški komplement
neg rd, rs	(sub rd, x0, rs)	dvojiški komplement
seqz rd, rs	(sltiu rd, rs, 1)	set if equal (to) zero
snez rd, rs	(sltu rd, x0, rs)	set if not equal (to) zero
sltz rd, rs	(slt rd, rs, x0)	set if less than zero
sgtz rd, rs	(slt rd, x0, rs)	set if greater than zero
...		

Pseudoinstruction	Base Instruction(s)	Meaning
`la rd, symbol`	`auipc rd, symbol[31:12]` `addi rd, rd, symbol[11:0]`	Load address
`l{b|h|w|d} rd, symbol`	`auipc rd, symbol[31:12]` `l{b|h|w|d} rd, symbol[11:0](rd)`	Load global
`s{b|h|w|d} rd, symbol, rt`	`auipc rt, symbol[31:12]` `s{b|h|w|d} rd, symbol[11:0](rt)`	Store global
`fl{w|d} rd, symbol, rt`	`auipc rt, symbol[31:12]` `fl{w|d} rd, symbol[11:0](rt)`	Floating-point load global
`fs{w|d} rd, symbol, rt`	`auipc rt, symbol[31:12]` `fs{w|d} rd, symbol[11:0](rt)`	Floating-point store global
`nop`	`addi x0, x0, 0`	No operation
`li rd, immediate`	*Myriad sequences*	Load immediate
`mv rd, rs`	`addi rd, rs, 0`	Copy register
`not rd, rs`	`xori rd, rs, -1`	One's complement
`neg rd, rs`	`sub rd, x0, rs`	Two's complement
`negw rd, rs`	`subw rd, x0, rs`	Two's complement word
`sext.w rd, rs`	`addiw rd, rs, 0`	Sign extend word
`seqz rd, rs`	`sltiu rd, rs, 1`	Set if = zero
`snez rd, rs`	`sltu rd, x0, rs`	Set if ≠ zero
`sltz rd, rs`	`slt rd, rs, x0`	Set if < zero
`sgtz rd, rs`	`slt rd, x0, rs`	Set if > zero
`beqz rs, offset`	`beq rs, x0, offset`	Branch if = zero
`bnez rs, offset`	`bne rs, x0, offset`	Branch if ≠ zero
`blez rs, offset`	`bge x0, rs, offset`	Branch if ≤ zero
`bgez rs, offset`	`bge rs, x0, offset`	Branch if ≥ zero
`bltz rs, offset`	`blt rs, x0, offset`	Branch if < zero
`bgtz rs, offset`	`blt x0, rs, offset`	Branch if > zero
`bgt rs, rt, offset`	`blt rt, rs, offset`	Branch if >
`ble rs, rt, offset`	`bge rt, rs, offset`	Branch if ≤
`bgtu rs, rt, offset`	`bltu rt, rs, offset`	Branch if >, unsigned
`bleu rs, rt, offset`	`bgeu rt, rs, offset`	Branch if ≤, unsigned
`j offset`	`jal x0, offset`	Jump
`jal offset`	`jal x1, offset`	Jump and link
`jr rs`	`jalr x0, rs, 0`	Jump register
`jalr rs`	`jalr x1, rs, 0`	Jump and link register
`ret`	`jalr x0, x1, 0`	Return from subroutine
`call offset`	`auipc x6, offset[31:12]` `jalr x1, x6, offset[11:0]`	Call far-away subroutine
`tail offset`	`auipc x6, offset[31:12]` `jalr x0, x6, offset[11:0]`	Tail call far-away subroutine

Psevdoukaz la je koristen za nalaganje naslova

- Če je naslov nizek, lahko naložimo vrednost v register takole:
  - lw rd, offset(rs), ali
  - lw rd, label(rs)

- Ne moremo pa naložiti naslova, večjega od 2^{11}-1 (2047)
- Poljuben naslov lahko naložimo v register s psevdoukazom la:

```
la rd, symbol
```

  - prevede se v:

```
auipc rd, symbol[31:12]
addi rd, rd, symbol[11:0]
```

Isti program za poljubne naslove:

```
        .data           # (na visokem naslovu, npr. 0x10000400)
A:      .word 5
B:      .word 0

        .text
        la x10, A
        lw x1, 0(x10)
        slli x2, x1, 3
        slli x3, x1, 1
        add x4, x2, x3
        la x10, B
        sw x4, 0(x10)
```