Procesorski sistemi v telekomunikacijah
Kodiranje števil, ukazi in izvajanje programa

Nepredznačena cela števila

- Zapis z n biti, možna števila $0..2^n-1$
- Kodiranje v dvojiškem zapisu
- Uteži mest so potence števila 2
- Število = vsota uteži neničelnih mest v zapisu

Bit	7	6	5	4	3	2	1	0
Utež	128	64	32	16	8	4	2	1

Uteži za dvojiški zapis nepredznačenih 8-bitnih števil

$10001111_2=$
$=128+8+4+2+1=$
$=143_{10}$

$18 = 2 \times 9 + 0$
$9 = 2 \times 4 + 1$
$4 = 2 \times 2 + 0$
$2 = 2 \times 1 + 0$
$1 = 2 \times 0 + 1$

$18_{10}=10010_2=$
$=00010010_2$

Predznačena cela števila - zapis z dvojiškim komplementom

- Zapis z n biti, možna števila $-2^{n-1}..2^{n-1}-1$
- Uteži mest so potence števila 2
- Utež najvišjega mesta (MSB) je negativna
- Število = vsota uteži neničelnih mest v zapisu

Bit	MSB 7	6	5	4	3	2	1	LSB 0
Utež	-128	64	32	16	8	4	2	1

Uteži za dvojiški zapis predznačenih 8-bitnih števil

- Negativna števila imajo MSB=1
- Operacija zamenjave predznaka = dvojiški komplement

$10001110_2=$
$=-128+8+4+2=-114_{10}$

$01001111_2=$
$=64+8+4+2+1=79_{10}$

Sprememba predznaka

- ... ali dvojiški komplement
- Izvedba:
  invertiraj vse bite, k dobljenemu številu prištej 1
- Primer za 8-bitni predznačen zapis (2^n=256)

$-115_{10} = 10001101_2$

Invertiraj bite
01110010_2
Prištej 1
$01110010_2 + 1$
Rezultat
$01110011_2 = 115_{10}$

$115_{10} = 01110011_2$

Invertiraj bite
10001100_2
Prištej 1
$10001100_2 + 1$
Rezultat
$10001101_2 = -115_{10}$

Zapis daljših števil v pomnilnik

- Zapis k-bitnih števil v n-bitne celice (k>n, k je deljiv z n)
- Razbijemo na skupine po n bitov
- Uporabimo k/n zaporednih celic za zapis
- Kaj pa vrstni red?

Zapis 16-bitnega števila v 8-bitne celice

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	MSW								LSW							

Little endian

Naslov	Vsebina
...	
a	LSW
a+1	MSW
...	

Uporablja ga
ARM7 (lahko)
LPC2138 (ARM7)
INTEL (npr i386)

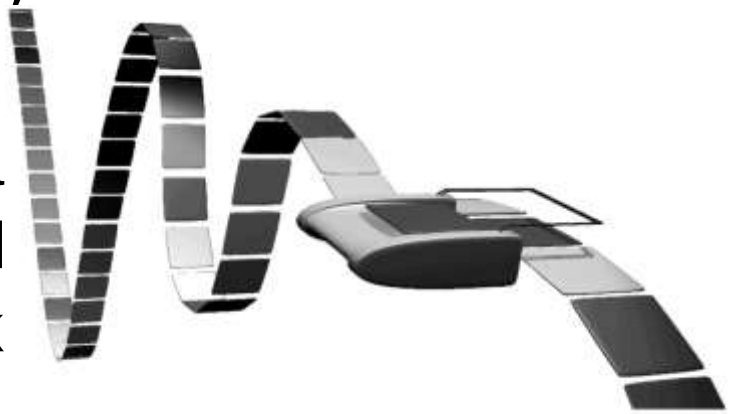
Big endian

Naslov	Vsebina
...	
a	MSW
a+1	LSW
...	

Uporablja ga
ARM7 (lahko)
Motorola 68k

Pomnilnik, kot ga vidi programer

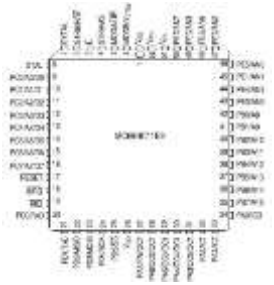
- Zbirka 2^m celic
- Vsaka celica ima naslov (m-bitno nepredzn. število)
- Množica vseh možnih naslovov = naslovni prostor
- Ena celica lahko hrani eno n-bitno število (običajno n=8)
- Tolmačenje tega števila (predznačeno/nepredznačeno) je prepuščeno programerju
- Posamezna celica lahko pripa
  a) bralnemu pomnilniku (ROM
  b) bralno/pisalnemu pomnilnik (RAM)



c) registru kake naprave

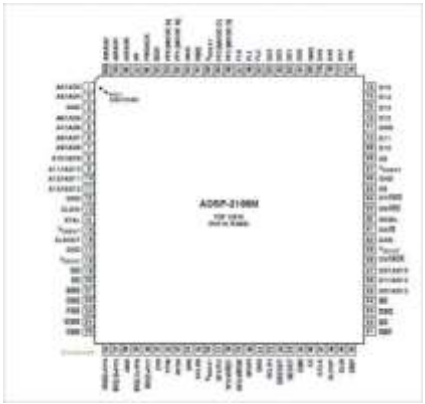
Primeri naslovnih prostorov (kot jih vidi programer)

68HC11
2^{16}=65536 8-bitnih celic



LPC2138 (ARM7)
2^{32}=4 294 967 296 8-bitnih celic

ADSP2100
2^{14}=16 384 24-bitnih celic za program
2^{14}=16 384 16-bitnih celic za podatke



Program in podatki v dveh ločenih pomnilnikih do katerih vodita dve ločeni skupini naslovnih in podatkovnih linij, ukazi lahko programski pomnilnik samo berejo –
Harvardska arhitektura

Ukazi lahko v programski pomnilnik tudi pišejo –
spremenjena Harvardska arhitektura

Kodiranje ukazov

- Večina ukazov je preveč zapletenih za zapis z 8 biti
- Običajno ukaz zaseda več zaporednih 8-bitnih celic
- Vsi ukazi nekega uP niso nujno enako dolgi

Trije ukazi 68HC11

Naslov	Vsebina (HEX)	
...		
a	86	LDAA #$A3
a+1	A3	
a+2	8B	ADDA #$12
a+3	12	
a+4	B7	STAA $2000
a+5	20	
a+6	00	
...		

Dva ukaza ARM7 (ARM način)

Naslov	Vsebina (HEX)	
...		
a	A3	MOV R0, #0xA3
a+1	00	
a+2	A0	
a+3	E3	
a+4	12	ADDS R0, R0, #0x12
a+5	00	
a+6	90	
a+7	E2	
...		

ARM7 - THUMB način delovanja – ukaz je dolg 16 bitov
ARM11 – THUMB-2 – 16-bitni in 32-bitni ukazi

Programski števec (PC), izvajanje ukaza

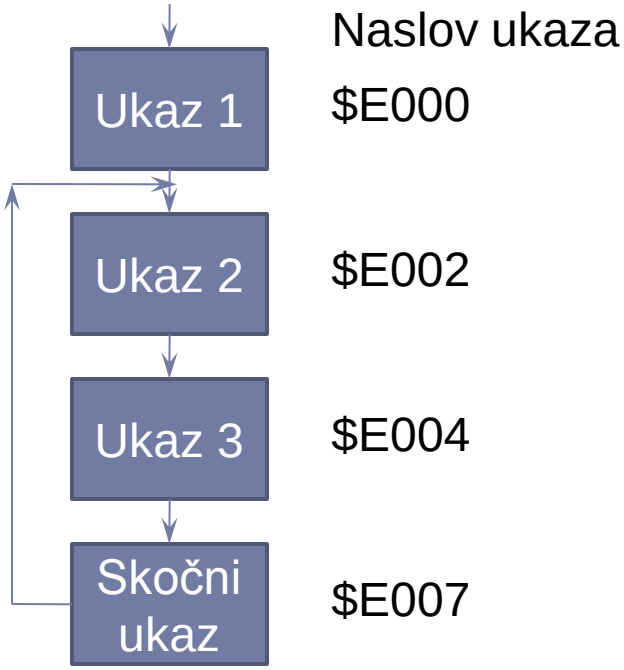
- 1. Preberi naslednji ukaz od naslova, ki ga hrani PC
- 2. Povečaj PC za dolžino ukaza
  PC sedaj hrani naslov naslednjega ukaza
- 3. Dekodiraj ukaz (ugotovi, kaj ukaz želi od procesorja)
- 4. Izvedi ukaz.
- 5. Vrni se na točko 1

- Ob izvajanju nekateri ukazi spreminjajo vrednost PC
- Če je sprememba brezpogojna ... **skočni ukaz**
- Če se sprememba PC zgodi le ob izpolnjenem pogoju
  ... **vejitveni ukaz**

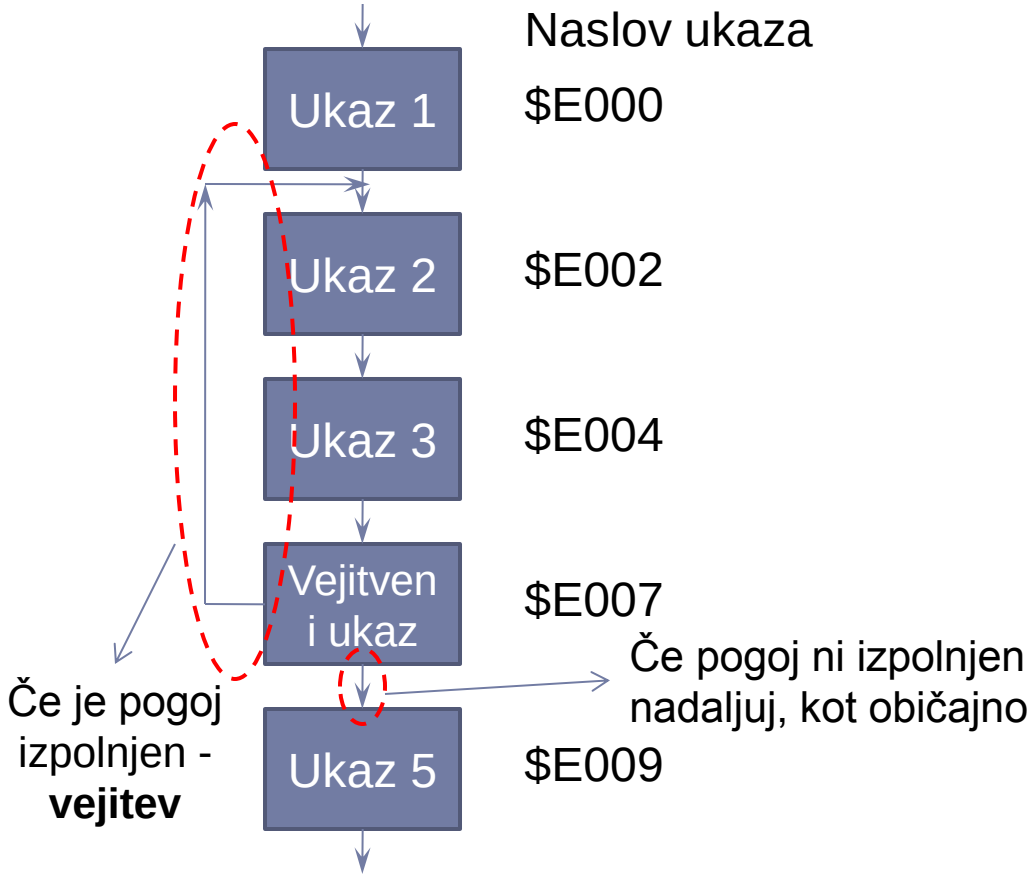
PC = Program Counter (Programski števec)

Skočni in vejitveni ukazi

Skočni ukaz spremeni PC brezpogojno in vsakič na enak način



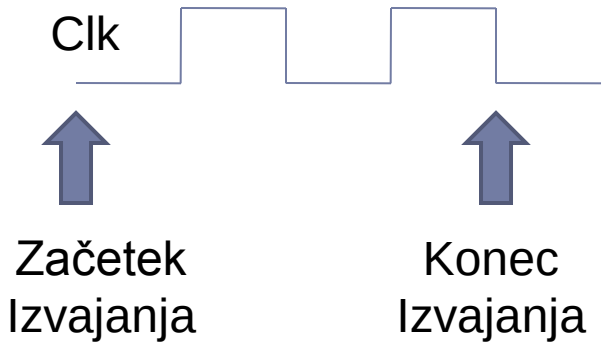
Vejitveni ukaz spremeni PC na en način, če je pripadajoč pogoj izpolnjen, in na drug način, če ta pogoj ni izpolnjen



Primer
Ukaz 'LDAA #$A3' v 68HC11

- **Zapiše konstanto $A3 v register A**
- **Začetno stanje**
  PC=$E000, A=?
- **Urin cikel 1:**
  Preberi $86 z naslova $E000, PC=PC+1
  Stanje po končanem ciklu 1
  PC=$E001, A=?
- **Urin cikel 2:**
  Preberi $A3 z naslova $E000, PC=PC+1
  Zapiši $A3 v register A
  Stanje po končanem ciklu 2
  PC=$E002, A=$A3

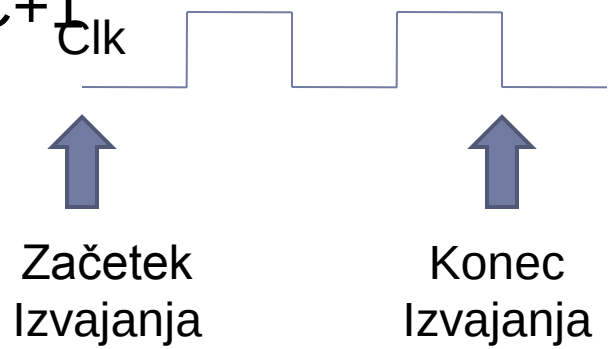
Naslov	Vsebina (HEX)
…	
$E000	86
$E001	A3
$E002	8B
$E003	12
$E004	B7
$E005	20
$E006	00
$E007	20
$E008	FE
…	



Primer
Ukaz 'ADDA #$12' v 68HC11

- **Prišteje konstanto $12 k vsebini registra A**
- **Začetno stanje**
  PC=$E002, A=$A3
- **Urin cikel 1:**
  Preberi $8B z naslova $E002, PC=PC+1
  Stanje po končanem ciklu 1
  PC=$E003, A=$A3
- **Urin cikel 2:**
  Preberi $12 z naslova $E003, PC=PC+1
  Prištej $12 k vsebini registra A
  Stanje po končanem ciklu 2
  PC=$E003, A=$B5

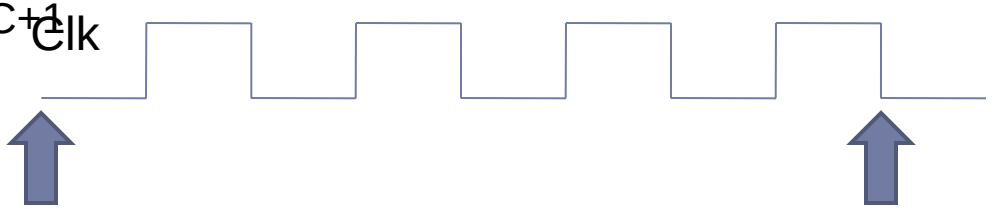
Naslov	Vsebina (HEX)
...	
$E000	86
$E001	A3
$E002	8B
$E003	12
$E004	B7
$E005	20
$E006	00
$E007	20
$E008	FE
...	



Primer
Ukaz ‘STAA $2000’ v 68HC11

- **Prepiše vsebino registra A v pomnilnik v celico z naslovom $2000**
- **Začetno stanje**
  PC=$E004, A=$B5, MEM($2000)=?
- **Urin cikel 1:**
  Preberi $B7 z naslova $E004, PC=PC+1
  Stanje po končanem ciklu 1
  PC=$E005, A=$B5, MEM($2000)=?
- **Urin cikel 2:**
  Preberi $20 z naslova $E005, PC=PC+1
  Stanje po končanem ciklu 2
  PC=$E006, A=$B5, MEM($2000)=?
- **Urin cikel 3:**
  Preberi $00 z naslova $E006, PC=PC+1
  Stanje po končanem ciklu 3
  PC=$E007, A=$B5, MEM($2000)=?
- **Urin cikel 4:**
  Zapiši vsebino registra A v pomnilnik na naslov $2000
  Stanje po končanem ciklu 4
  PC=$E007, A=$B5, MEM($2000)=$B5

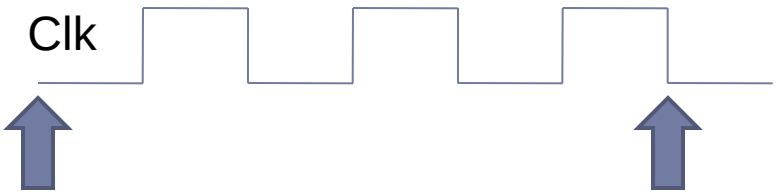
Naslov	Vsebina (HEX)
…	
$E000	86
$E001	A3
$E002	8B
$E003	12
$E004	B7
$E005	20
$E006	00
$E007	20
$E008	FE
…	



Primer
Ukaz 'BRA $FE' v 68HC11

- **Zmanjša PC za 2 (brezpogojni skok)**
- **Začetno stanje**
  PC=$E007
- **Urin cikel 1:**
  Preberi $20 z naslova $E007, PC=PC+1
  Stanje po končanem ciklu 1
  PC=$E008
- **Urin cikel 2:**
  Preberi $FE z naslova $E008, PC=PC+1
  Stanje po končanem ciklu 2
  PC=$E009
- **Urin cikel 3:**
  Prištej $FE (predzn. zapis za -2) k vsebini PC
  Stanje po končanem ciklu 3
  PC=$E007

Naslov	Vsebina (HEX)
…	
$E000	86
$E001	A3
$E002	8B
$E003	12
$E004	B7
$E005	20
$E006	00
$E007	20
$E008	FE
…	



Primer
Potek še malo drugače

Ukaz								
1	Naloži $86 (LDAA #)	Naloži $A3, Piši $A3 v A						
2			Naloži $8B (ADDA #)	Naloži $12, in prištej k A				
3					Naloži $B7 (STAA)	Naloži $20 (MSW $2000)	Naloži $00 (LSW $2000)	Zapiši A na naslov $2000
Urin cikel	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
PC (začetek, konec cikla)	$E000 $E001	$E001 $E002	$E002 $E003	$E003 $E004	$E004 $E005	$E005 $E006	$E006 $E007	$E007 $E007
A (začetek, konec cikla)	? ?	? $A3	$A3 $A3	$A3 $B5	$B5 $B5	$B5 $B5	$B5 $B5	$B5 $B5
MEM($2000) (konec cikla)	?	?	?	?	?	?	?	$B5

čas

Cevovod (pipeline) in izvajanje ukazov

- Primer: ARM7, cevovod dolžine 3
- Kot tekoči trak v tovarni – vzporedno delo na večih izdelkih.
- Hkrati se dogaja
  nalaganje ukaza **i-2**,
  dekodiranje ukaza **i-1** in
  izvajanje ukaza **i**

Ukaz						
1	naloži 1	dekodiraj 1	izvedi 1			
2		naloži 2	dekodiraj 2	izvedi 2		
3			naloži 3	dekodiraj 3	izvedi 3	
…				…	…	…
Urin cikel	1	2	3	4	5	…

čas

- V povprečju naloži/dekodira/izvrši **1 ukaz / cikel ure**

Kaj potrebujemo za 1ukaz/cikel pri prej omenjenem cevovodu?

- **1. Mikroprocesor mora ukaz naložiti v 1 ciklu ure.**
- **Kako lahko ARM7 naloži 1 ukaz/cikel**, če je ukaz dolg 32 bitov, ena celica pomnilnika pa hrani le 8 bitov?
  ARM7 v enem ciklu ure prebere 32 bitov iz pomnilnika, programerju pa jih predoči kot 4 celice po 8 bitov.
  Posledica – pomnilniški modul mora biti širok 32 bitov, Ena celica tega modula = 4 celice naslovnega prostora.

- **2. Ukaz se mora izvršiti v enem ciklu ure.**
- Posledica: ukazi morajo biti preprosti.

CISC

- Complex Instruction Set Computer
- Lastnosti CISC mikroprocesorjev
  - Veliko različnih ukazov, veliko kombinacij naslavljanja in operandov (ortogonalnost), zapleteni ukazi
  - Dolgotrajno izvajanje posameznega ukaza
  - Bolj zapleten mikroprocesor, večja poraba.
  - Lažje izdelati učinkovit prevajalnik za višji jezik (npr. C)
  - Programi v strojnem jeziku so krajši
  - Primer: PDP-11 (1970), Motorola 68000 (1979) Intel x86 družina (od 8086 naprej)

RISC

- Reduced Instruction Set Computer
- Lastnosti RISC mikroprocesorjev
  - Malo ukazov, ukazi so preprosti
  - Izvajanje posameznega ukaza je hitro
  - Preprosta izvedba, majhen po velikosti, majhna poraba
  - Učinkovit prevajalnik za višji jezik (npr. C) je bolj zapleten
  - Programi v strojnem jeziku so daljši (pribl. 5x)
  - Prvi RISC računalnik: CDC 6600 (1964)
  - SPARC (1987) – Sun Microsystems
  - R2000 (1986) – MIPS Computer Systems
    MIPS R3000A v Sony PlayStation (1994)
  - ARM jedro je RISC

RISC preoblečen v CISC

- Primer: Intel od Pentiuma naprej

- Danes meja med RISC in CISC ni več tako ostra

Kaj pa če ukaz želi pisati v pomnilnik?

- Hkrati želimo izvesti: **branje pomnilnika** (zaradi nalaganja i-tega ukaza) in
  pisanje v pomnilnik (zaradi izvajanja i-2. ukaza).
- To ne gre!!! (eno vodilo = en dostop do pomnilnika na urin cikel).
- **Posledica**: med pisanjem v pomnilnik ne moremo nalagati novega ukaza.
- **Možna rešitev: (**spremenjena) Harvardska arhitektura
  - dva ločena pomnilnika - en za program in en za podatke,
  - hkrati lahko poteka nalaganje ukaza iz programskega pomnilnika in pisanje podatka v podatkovni pomnilnik

Pisanje v pomnilnik (1. ukaz) Ne moremo nalagati 3. ukaza

Ukaz							
1	naloži 1	dekodiraj 1	izvedi 1				
2		naloži 2	dekodiraj 2	izvedi 2			
3				naloži 3	dekodiraj 3	izvedi 3	
…					…	…	…
Urin cikel	1	2	3	4	5	6	…

ARM7 je še malo bolj zapleten!

- **Primer:** pisanje v pomnilnik (1. ukaz na sliki) zahteva predpripravo - **izračun naslova**, kamor bomo pisali.
- Izvajanje ukaza STR zahteva 2 urina cikla (računanje+pisanje).
- **Med računanjem** naslova **ne moremo dekodirati ali izvajati drugih ukazov**, ker so linije za podatke (datapath) zasedene.
- **Med pisanjem** v pomnilnik **ne moremo nalagati ukaza**.

zakasnitev dekodiranja 2. in 3. ukaza (**pipeline stall**), ker prvi ukaz zaseda podatkovne linije

Zakasnitev nalaganja 4. ukaza, ker 1. ukaz piše v pomnilnik

Ukaz							
1	naloži 1	dekodiraj 1	računaj 1	izvedi 1			
2		naloži 2		dekodiraj 2	izvedi 2		
3			naloži 3		dekodiraj 3	izvedi 3	
...					naloži 4	dekodiraj 4	izvedi 4
					...	...	...
Urin cikel	1	2	3	4	5	6	...

Skočni in vejitveni ukazi v cevovodu

- Ob brezpogojnem skoku in ob vejitvi se zgodi neobičajna sprememba PC

Naslov	Ukaz
A000	Ukaz 1: skoči na naslov B000 - naslednji se izvede ukaz 4
A004	Ukaz 2: ... (mi pa smo že naložili ukaza 2 in 3 ☹)
A008	Ukaz 3: ...
...	...
B000	Ukaz 4: ...
B004	Ukaz 5: ...

Nekoristen **mehurček** (**bubble**) v cevovodu

Teh korakov procesor ne izvaja

Ukaz							
1	naloži 1	dekodiraj 1	izvedi 1				
2		naloži 2	dekodiraj 2	izvedi 2			
3			naloži 3	dekodiraj 3	izvedi 3		
...				naloži 4	dekodiraj 4	izvedi 4	
					naloži 5	dekodiraj 5	izvedi 5
						...	...
Urin cikel	1	2	3	4	5	6	...

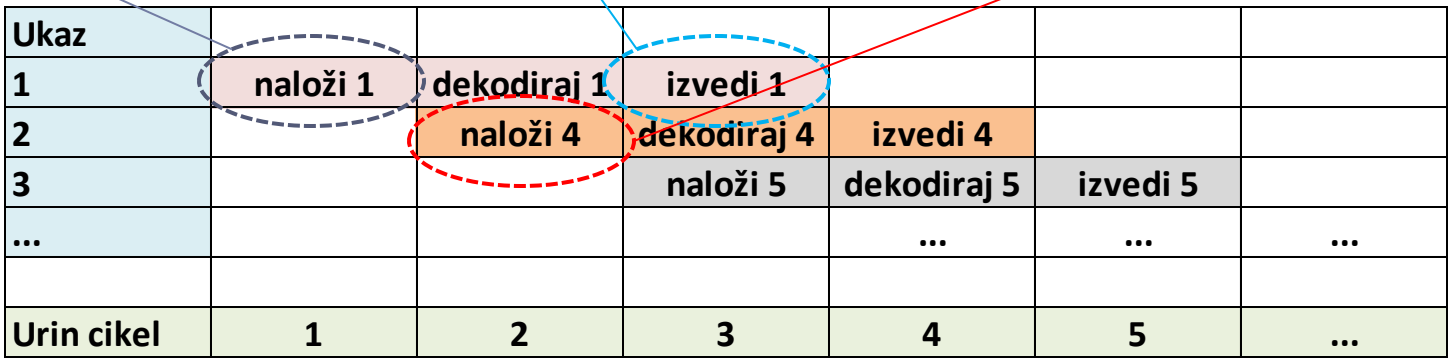
Pojav mehurčka

Mehurček izgine

Možna rešitev – napoved vejitve (branch prediction)

- Že ob nalaganju vejitvenega ukaza procesor poskuša napovedati, ali bo pogoj za vejitev v času izvajanja ukaza izpolnjen.
- Če je napoved "pogoj bo izpolnjen", kot naslednjega naložimo tisti ukaz, ki se bo izvedel po končani vejitvi.
- Kazen za napačno napoved (branch misprediction) je mehurček v cevovodu.
- 100% natančna napoved ni zmeraj mogoča.

Tukaj napovemo, de se bo **tukaj** zgodila vejitev, zato **tukaj** naložimo ukaz 4



Kaj pa daljši cevovodi?

- **ARM9 – cevovod dolžine 5**
  1. naloži ukaz
  2. dekodiraj ukaz
  3. računaj z operandi ali izračunaj naslov v pomnilniku
  4. dostopaj do pomnilnika (beri/piši)
  5. zapiši rezultat 3. oz 4. koraka v ustrezen register
- Pogost pri RISC mikroprocesorjih

- Nekateri ukazi v 4. koraku ne naredijo ničesar
  Npr. ARM ukaz **ADD R3, R1, R2**
  Sešteje vsebini registrov R1 in R2 in shrani rezultat v registru R3.
- Nekateri ukazi v 5. koraku ne naredijo ničesar
  Npr. ukazi za pisanje v pomnilnik - **STR**

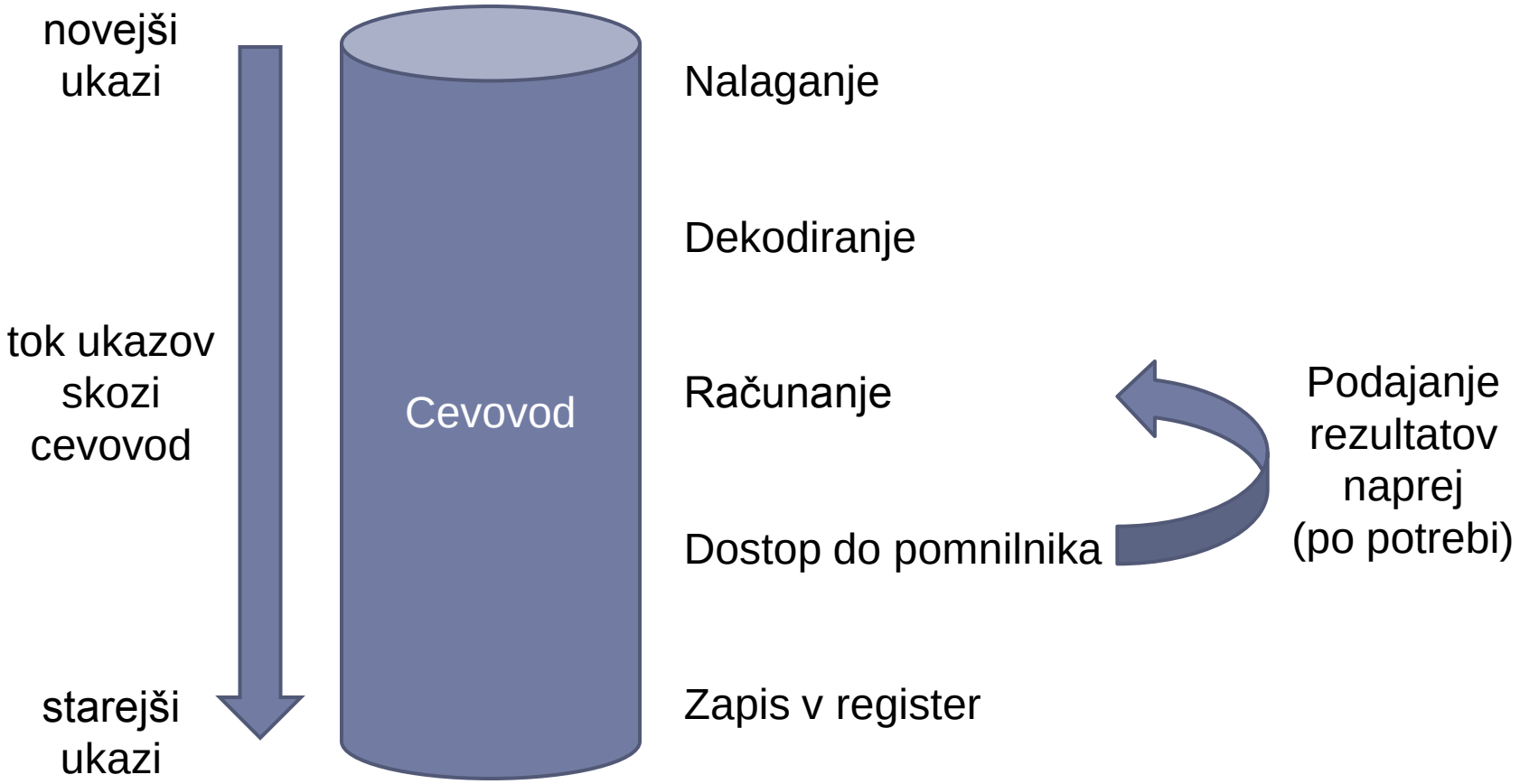
Daljši cevovodi
Medsebojna odvisnost ukazov

- Izvajanje ukaza razdeljeno v 3 korake (računanje, dostop do pomnilnika, zapis v register)
- Kaj če drugi ukaz v 1. koraku svojega izvajanja zahteva rezultate prvega ukaza, ki so na voljo šele v 3. koraku njegovega izvajanja
- Možna rešitev: ne uporabljati neugodnih kombinacij ukazov
- Potreben je dober programer oz. dober prevajalnik

Zakasnitev izvajanja 2. ukaza zaradi odvisnosti od rezultatov prvega ukaza

Ukaz								
1	naloži 1	dekodiraj 1	računaj 1	pomn. 1	zapis 1			
2		naloži 2	dekodiraj 2			računaj 2	pomn. 2	zapis 2
…			…	…	…	…	…	…
Urin cikel	1	2	3	4	5	6	7	…

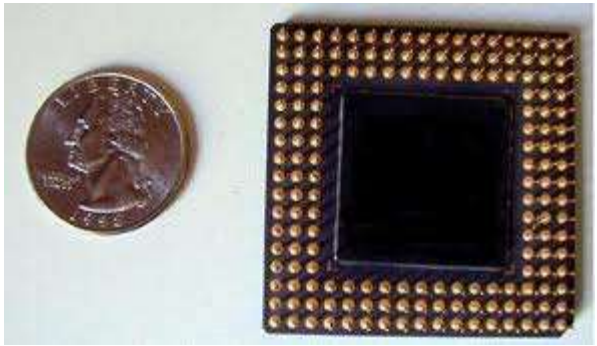
Strojna rešitev odvisnosti med ukazi
Podajanje rezultatov naprej (forwarding)



In kaj na to pravi Intel?

- 80486 je že imel cevovod (dolžine 5)
- CISC ukazi 80486 so prekompleksni, da bi bil cevovod učinkovit.
- Novejši procesorji prevajajo CISC ukaze v zaporedje enega ali več internih RISC ukazov.
- Jedro je RISC od Pentiuma naprej.
- Pentium 4: cevovod dolžine 20
- Core i7: cevovod dolžine 14

80486 (1989)



Pentium (1993)

Superskalarni mikroprocesorji

- Prebere več ukazov v enem urinem ciklu
- Preveri odvisnost ukazov od rezultatov prejšnjih ukazov
- Če je mogoče, izvede več ukazov vzporedno
- Procesor vsebuje več enakih funkcionalnih enot (npr. seštevalnikov)
- Običajno v kombinaciji s cevovodnim izvajanjem (ni pa nujno)
- Intel superskalaren od Pentiuma naprej (1993)
- Ni preveč učinkovito – naložimo 8 ukazov, v povprečju pa jih izvedemo le 2,1/cikel

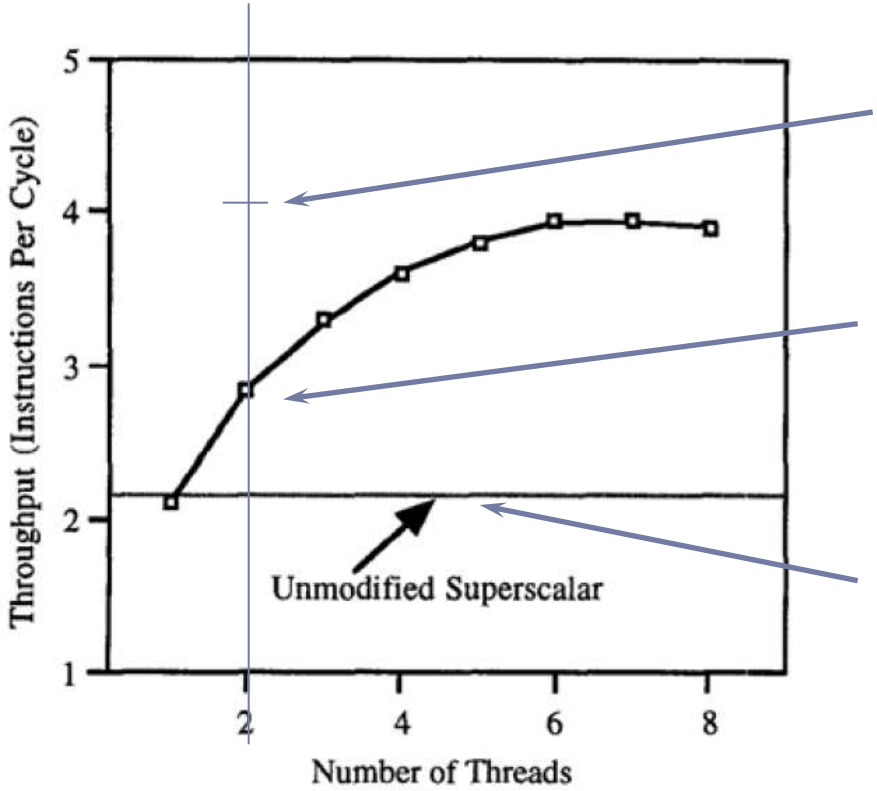
Ukaz								
1	naloži 1	dekodiraj 1	računaj 1	pomn. 1	zapis 1			
2	naloži 2	dekodiraj 2	računaj 2	pomn. 2	zapis 2			
3		naloži 3	dekodiraj 3	računaj 3	pomn. 3	zapis 3		
4		naloži 4	dekodiraj 4	računaj 4	pomn. 4	zapis 4		
5			naloži 5	dekodiraj 5	računaj 5	pomn. 5	zapis 5	
6			naloži 6	dekodiraj 6	računaj 6	pomn. 6	zapis 6	
…				…	…	…	…	…
Urin cikel	1	2	3	4	5	6	7	…

Hkratna večnitnost (Simultaneous Multithreading, SMT)

- Če nalagamo 8 ukazov hkrati, mora procesor imeti 8 enakih funkcionalnih enot (npr. seštevalnikov), da lahko vzporedno izvede 8 ukazov.
- Povprečno 2.1 ukaza na cikel - 6 od 8 funkcionalnih enot neizkoriščenih ☹
- Ideja: enote lahko izkoristimo za vzporedno izvajanje večih programov
  hkratna večnitnost
- 1 nit = 1 zaporedje ukazov = 1 program
- Potrebujemo več programskih števcev / procesor (1 števec / nit)
- Izvajanje vsake niti ponavadi poteka superskalarno.
- Pospeši se izvajanje programov, ki so že v osnovi načrtovani za vzporedno izvajanje (sestavljeni iz več niti, ki se lahko izvajajo vzporedno).
- Intel: Hyper-Threading (HT)
  XEON (2002), Pentium 4 (2002) ...
  Core i7 (2008) ... 2 niti/jedro (4 jedra ... 8 niti)

Primer: Kako dobro deluje SMT

- Nalaganje do 8 ukazov na cikel (superskalarnost)



2) SMT - pričakujemo 2.1 ukaza na cikel x 2 niti = 4.2 ukaze na cikel

3) SMT - dobimo malo manj kot 3 ukaze na cikel (v povprečju)

1) Navaden superskalaren procesor naloži 8 ukazov naenkrat, izvrši pa v povprečju 2.1 ukaza na cikel

Tullsen et. al., 23rd Annual Symposium on Computer Architecture, (c)1996 IEEE