Konstrukcija kompilatora

— Optimizacija koda —

Milena Vujošević Janičić

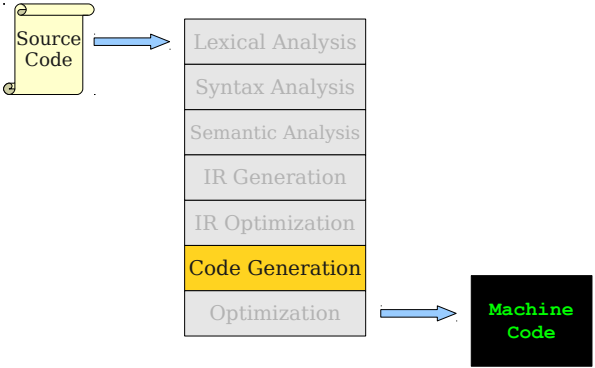
www.matf.bg.ac.rs/~milena

Matematički fakultet, Univerzitet u Beogradu

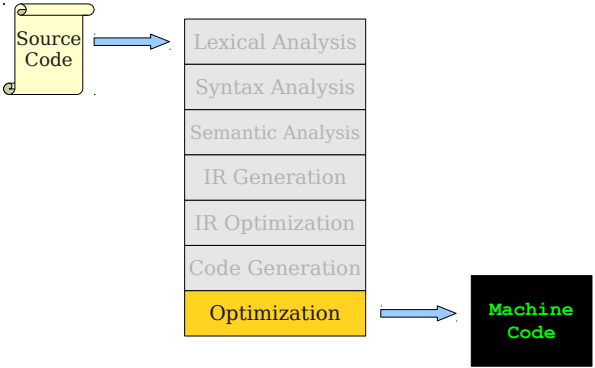
Pregled

Pregled

Where We Are


Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Machine Code

Where We Are


Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Machine Code

Finalna optimizacija koda

- Cilj: Optimizovati generisani kod korišćenjem mašinski zavisnih osobina koje nisu vidljive na IR nivou
- Kritičan korak većine kompajlera, ali obično veoma neuredan:
  - Tehnike koje se razviju za jednu mašinu mogu da budu potpuni nekorisne za drugu
  - Tehnike koje se razviju za jedan jezik mogu da budu potpuno nekorisne za drugi
- Razmotrićemo optimizaciju raspoređivanja instrukcija
- Postoje i razne druge optimizacije koda, npr optimizacije sa ciljem boljeg upravljanja kešom

Pregled

Processor Pipelines

Processor Pipelines

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

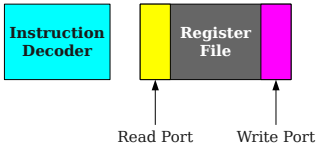
Processor Pipelines


```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

Processor Pipelines

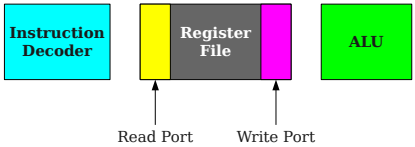

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

Processor Pipelines



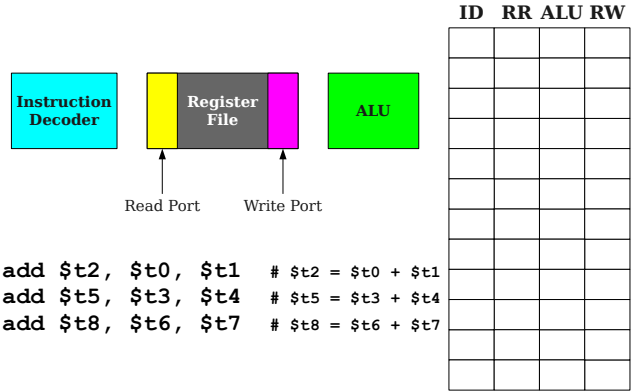

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

Processor Pipelines



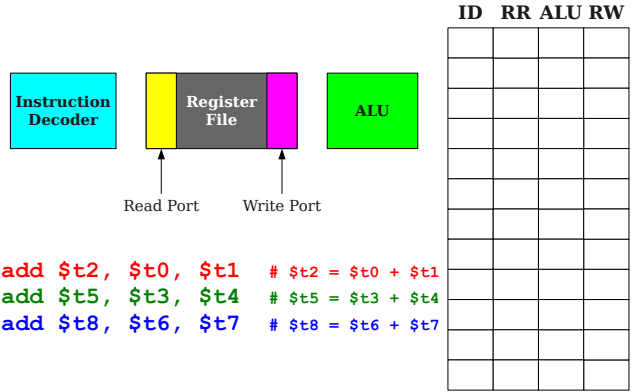

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

Processor Pipelines



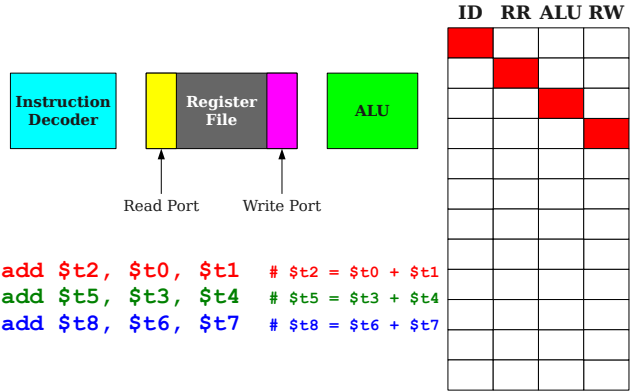

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

Processor Pipelines



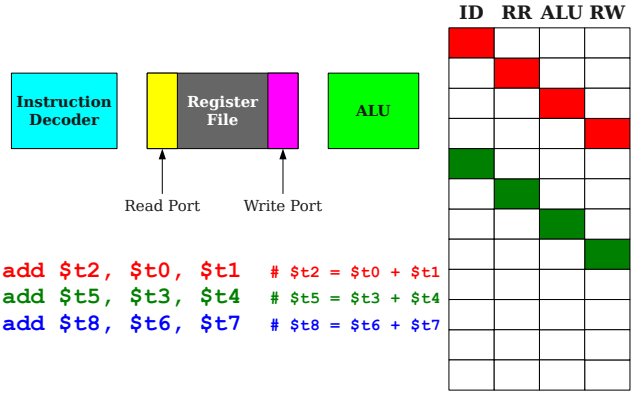

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

Processor Pipelines



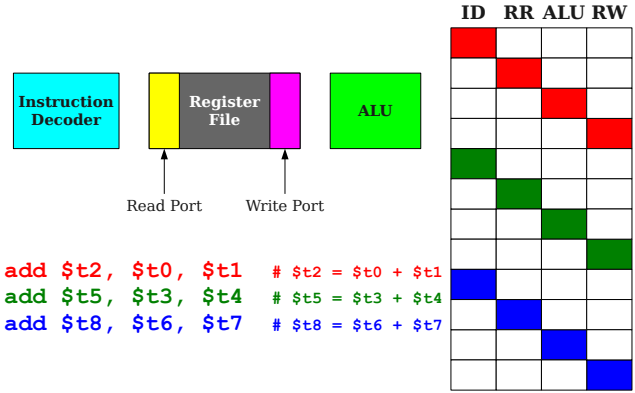

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

Processor Pipelines



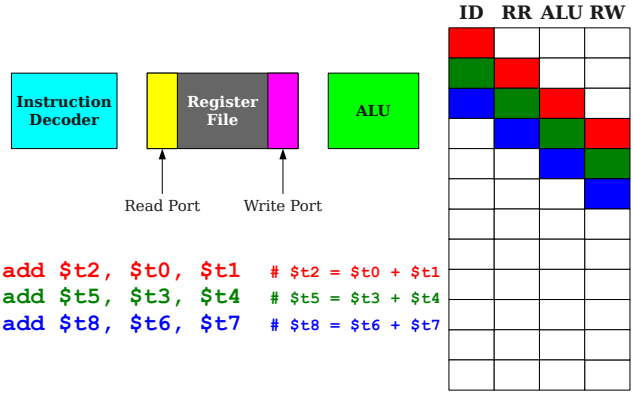

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

Processor Pipelines




```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

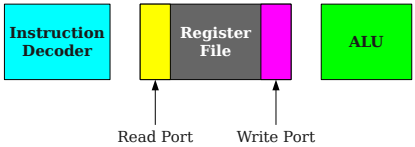
Processor Pipelines



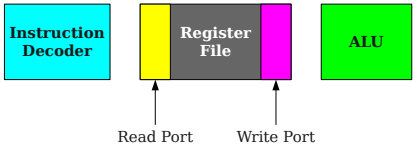

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

Pipeline Hazards

Pipeline Hazards

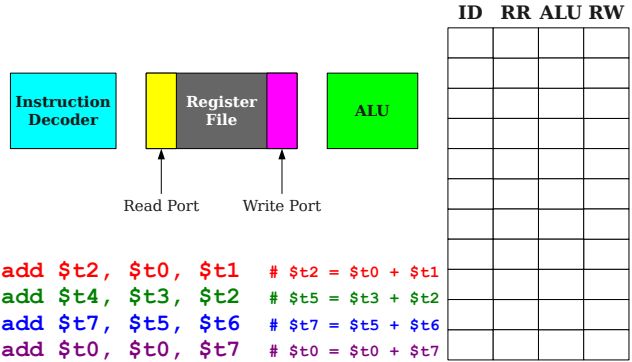


Pipeline Hazards



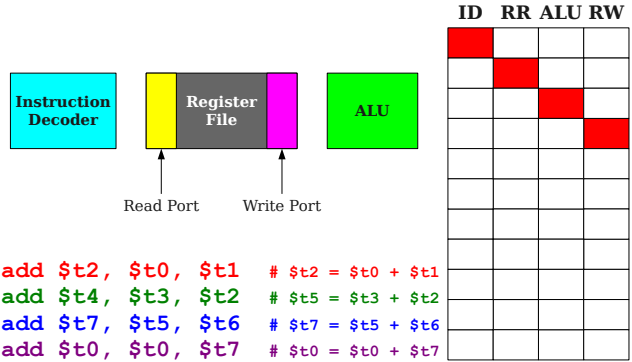

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

Pipeline Hazards



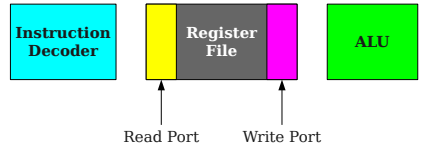

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

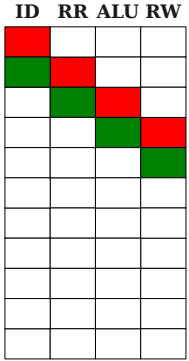
Pipeline Hazards




```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

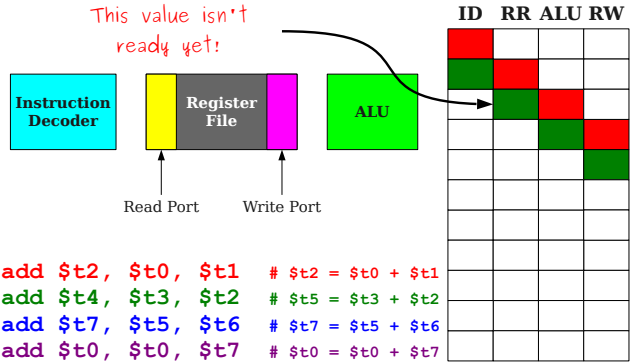
Pipeline Hazards





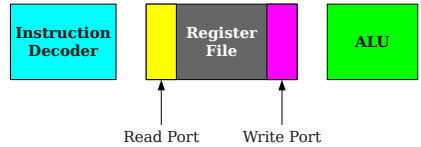

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

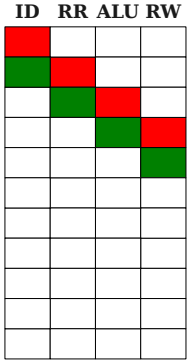
Pipeline Hazards




```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

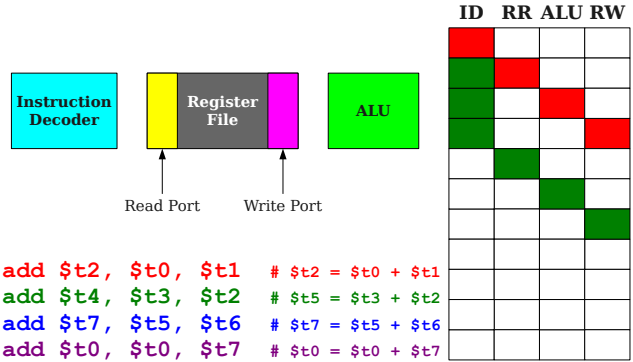
Pipeline Hazards





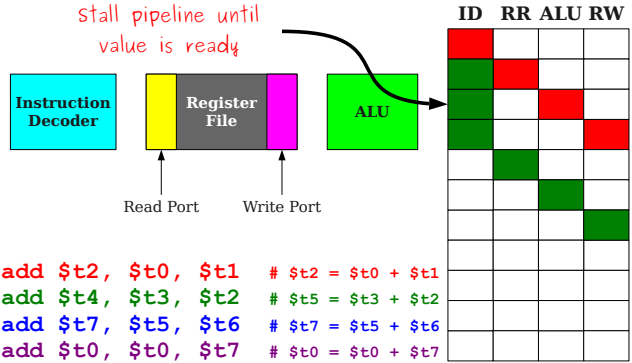

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

Pipeline Hazards



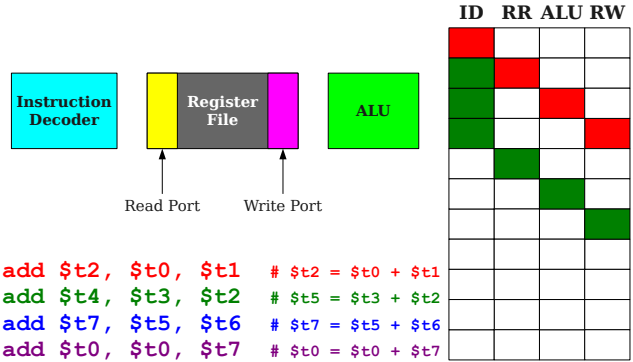

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

Pipeline Hazards



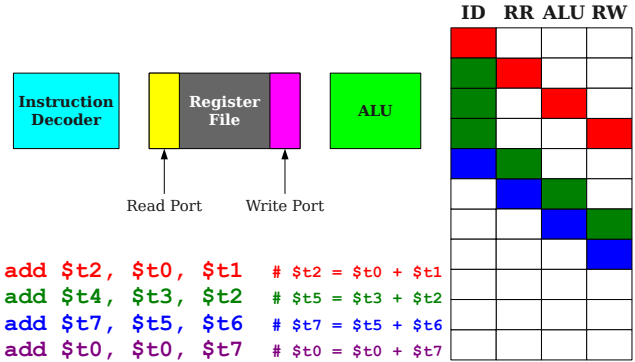

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

Pipeline Hazards



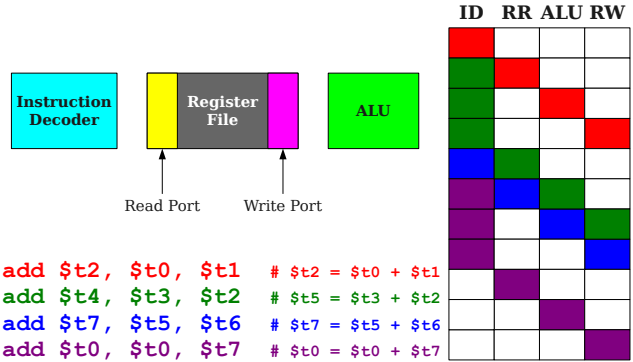

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

Pipeline Hazards



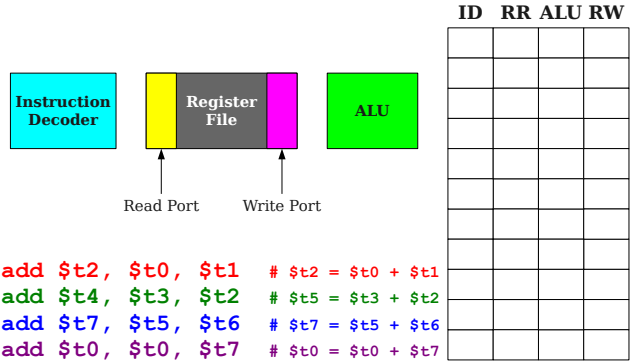

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

Pipeline Hazards



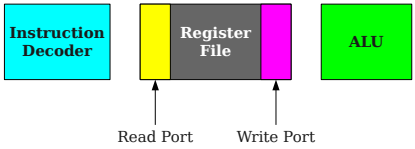

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

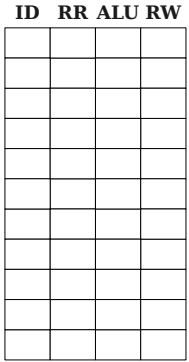
Pipeline Hazards




```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

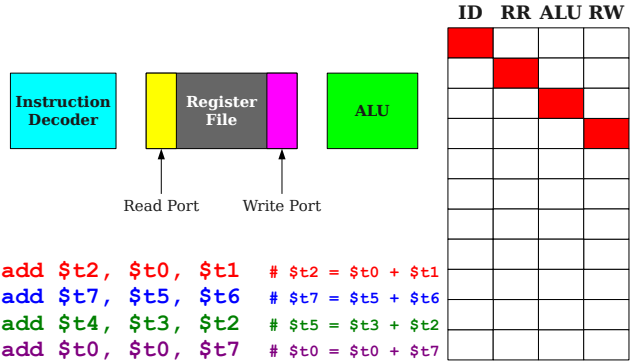
Pipeline Hazards





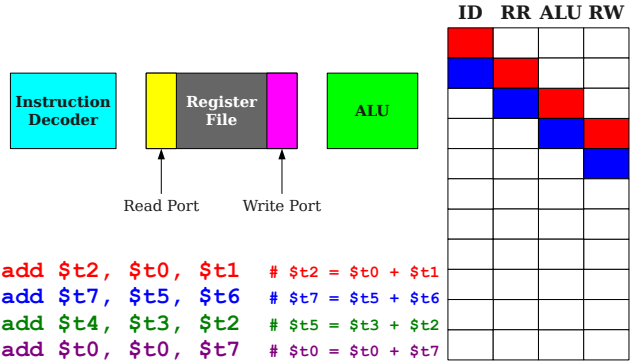

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

Pipeline Hazards



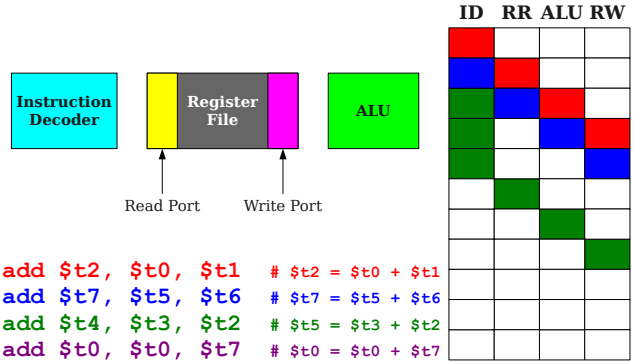

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

Pipeline Hazards



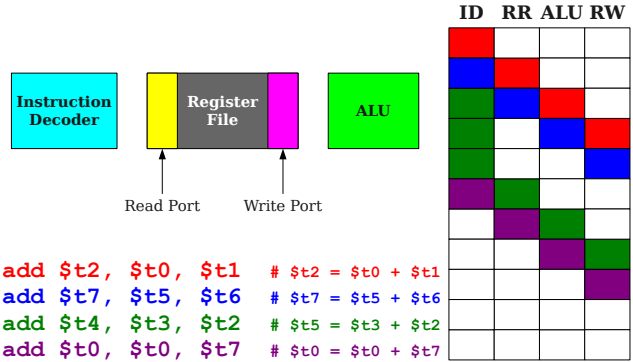

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

Pipeline Hazards



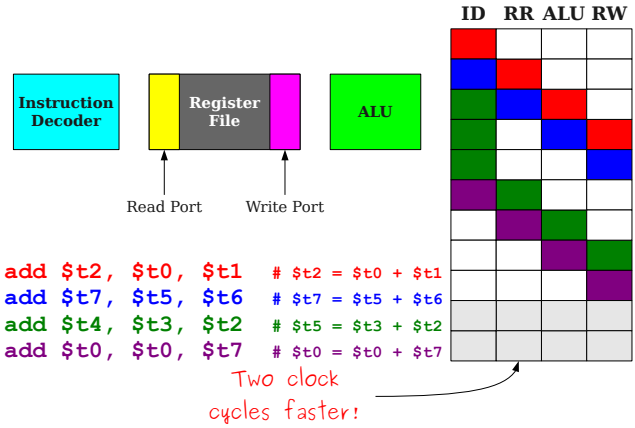

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

Pipeline Hazards




```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

Pipeline Hazards



Raspoređivanje instrukcija

- Zbog procesorskog *pipelining*-a, redosled u kojem se instrukcije izvršavaju može da utiče na performanse
- **Raspoređivanje instrukcija je pravljenje rasporeda instrukcija sa ciljem da se poboljšaju performanse**
- Svi dobri kompajleri imaju neku vrstu podrške za raspoređivanje instrukcija

Zavisnost među podacima

- Zavisnost među podacima u mašinskom kodu je skup instrukcija čije ponašanje zavisi jedno od druge
- Intuitivno, skup instrukcija koje ne mogu da se poređaju na drugačiji način
- Postoje tri vrste zavisnosti: čitanje nakon pisanja, pisanje nakon čitanja, pisanje nakon pisanja

Finding Data Dependencies

```
t0 = t1 + t2
t1 = t0 + t1
t3 = t2 + t4
t0 = t1 + t2
t5 = t3 + t4
t6 = t2 + t3
```

Finding Data Dependencies

```
t0 = t1 + t2

t1 = t0 + t1

t3 = t2 + t4

t0 = t1 + t2

t5 = t3 + t4

t6 = t2 + t7
```

Finding Data Dependencies

```
t0 = t1 + t2
```

```
t1 = t0 + t1
```

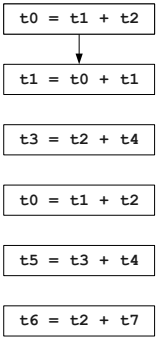
```
t3 = t2 + t4
```

```
t0 = t1 + t2
```

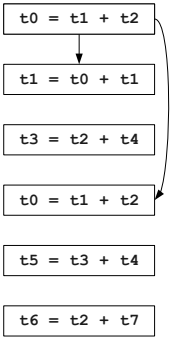
```
t5 = t3 + t4
```

```
t6 = t2 + t7
```

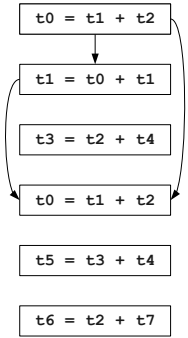
Finding Data Dependencies



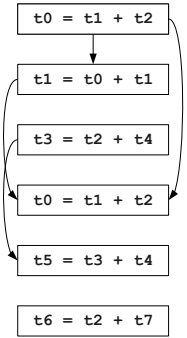
Finding Data Dependencies



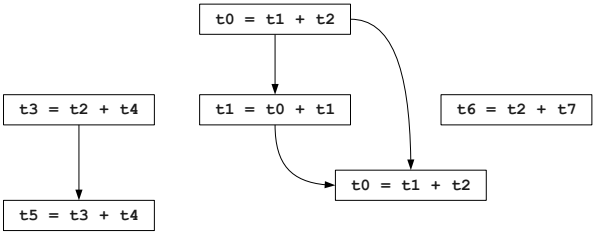
Finding Data Dependencies



Finding Data Dependencies



Finding Data Dependencies

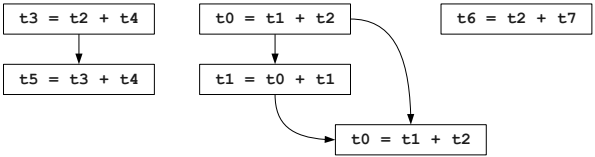


Graf zavisnosti podataka

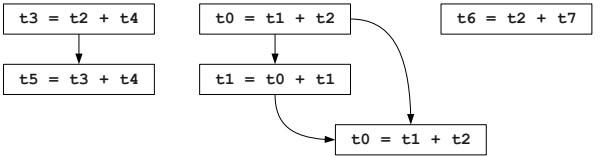
- Graf koji prikazuje zavisnosti podataka u okviru osnovnog bloka naziva se graf zavisnosti podataka
- To je direktan aciklični graf. Direktan, jer uvek jedna instrukcija zavisi od druge. Acikličan, jer nisu moguće ciklične zavisnosti.
- Mogu se rasporediti instrukcije u okviru osnovnog bloka u bilo kom redosledu sve dok se ne raspodele instrukcije tako da neka instrukcija prethodi svom roditelju
- Ideja: **napravi topološko sortiranje zavisnosti podataka i poređaj instrukcije u tom redosledu**

Instruction Scheduling

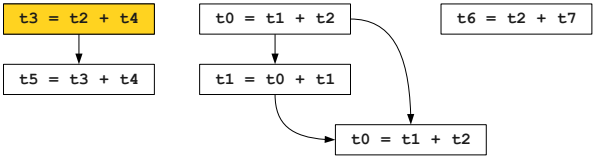
Instruction Scheduling



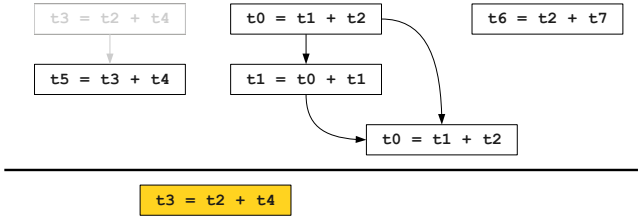
Instruction Scheduling



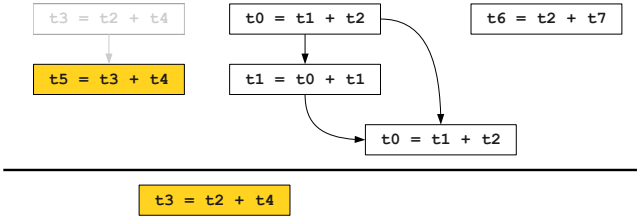
Instruction Scheduling



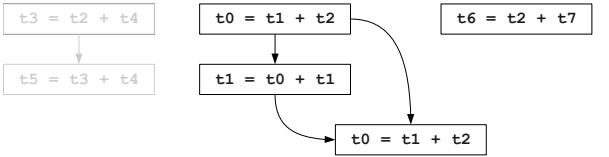
Instruction Scheduling

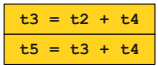


Instruction Scheduling

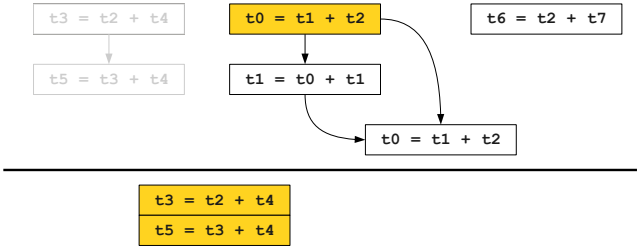


Instruction Scheduling

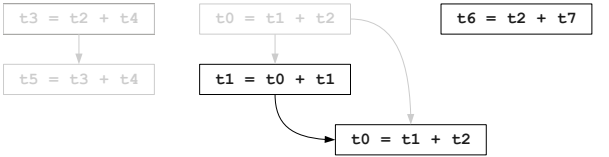


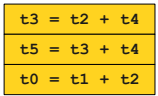


Instruction Scheduling

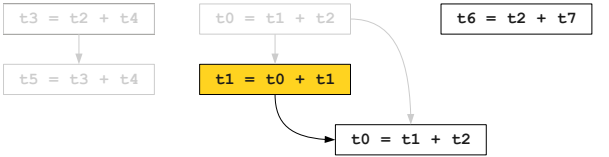


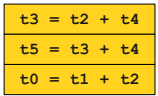
Instruction Scheduling



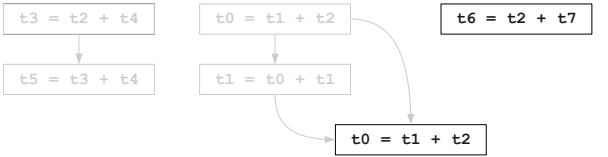


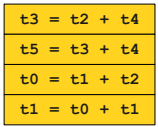
Instruction Scheduling



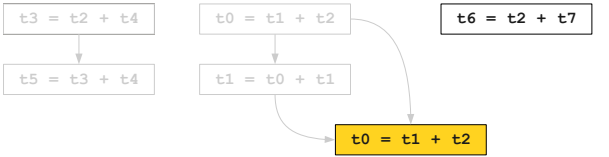


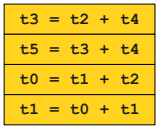
Instruction Scheduling



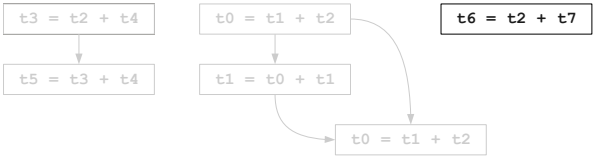


Instruction Scheduling



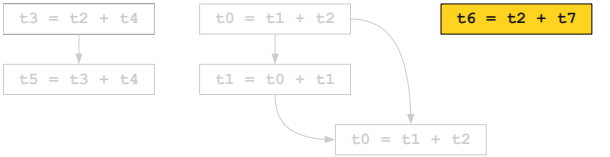


Instruction Scheduling



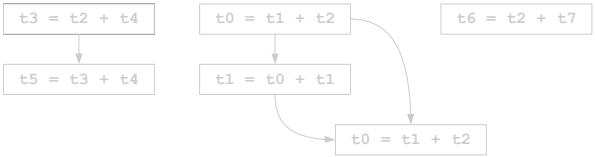

```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
```

Instruction Scheduling



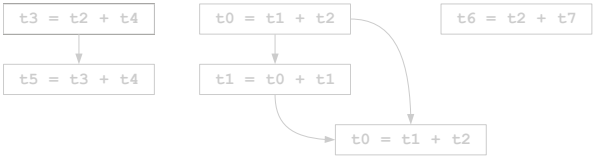
```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
```

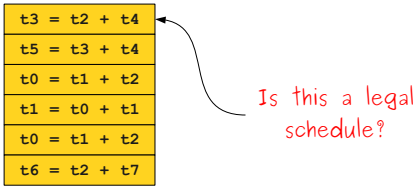
Instruction Scheduling



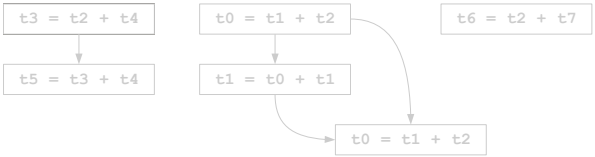

```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

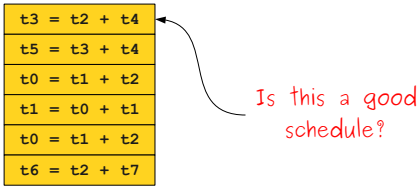
Instruction Scheduling



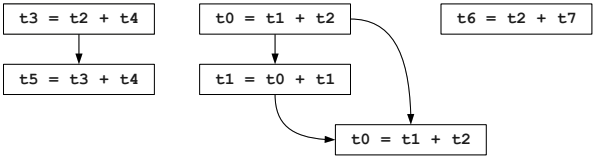


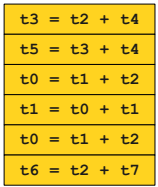
Instruction Scheduling



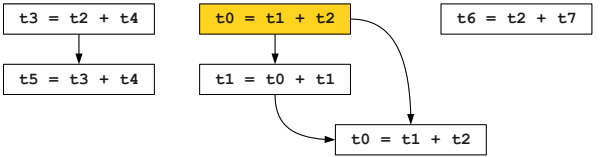


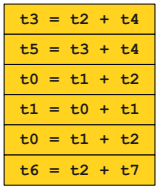
Instruction Scheduling



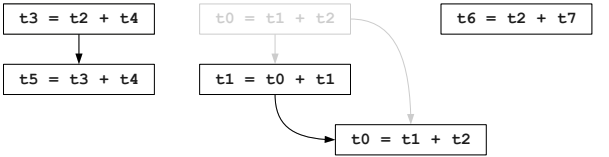


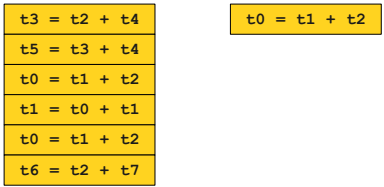
Instruction Scheduling



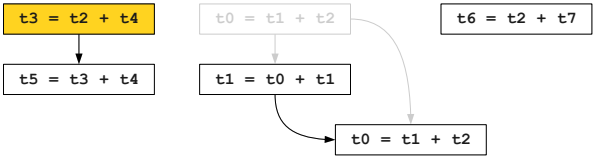


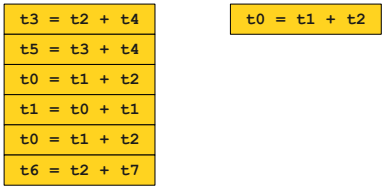
Instruction Scheduling



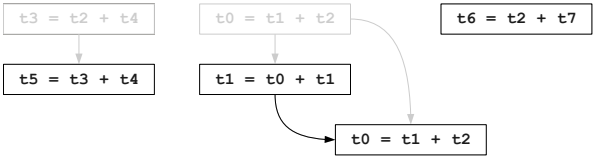


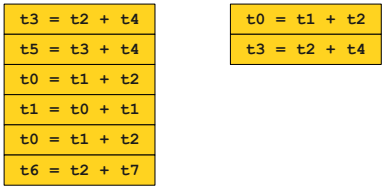
Instruction Scheduling



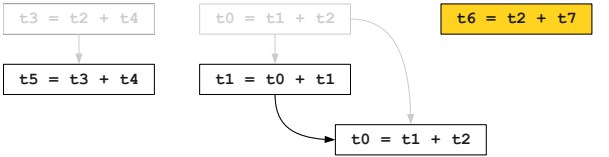


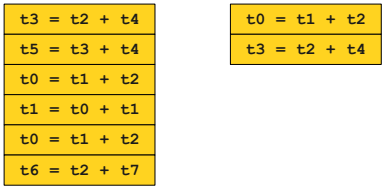
Instruction Scheduling



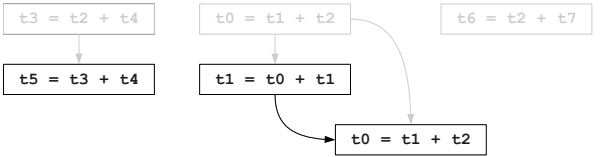


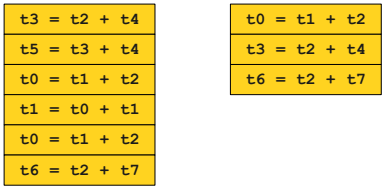
Instruction Scheduling



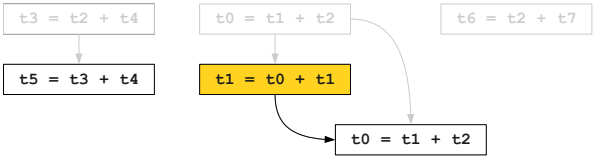


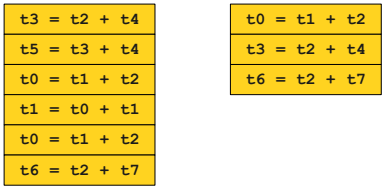
Instruction Scheduling



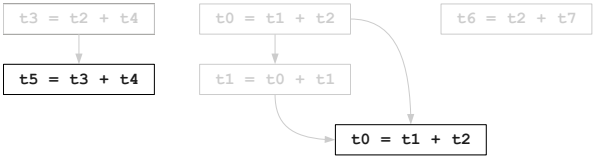


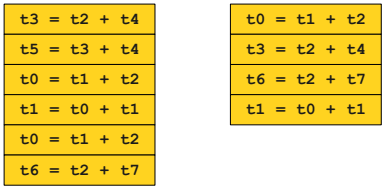
Instruction Scheduling



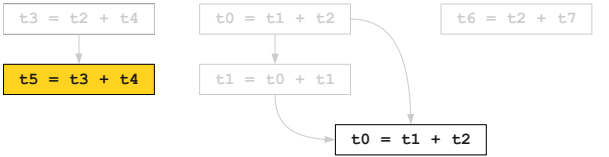


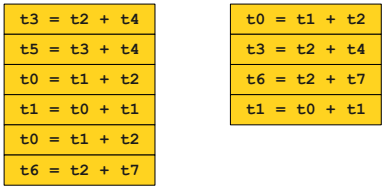
Instruction Scheduling



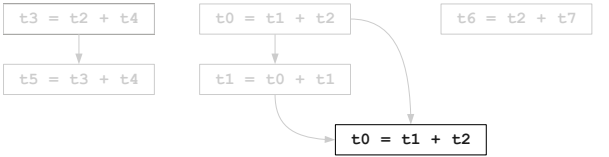


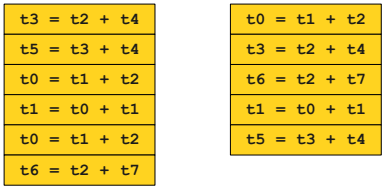
Instruction Scheduling



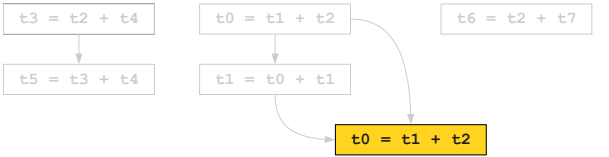


Instruction Scheduling



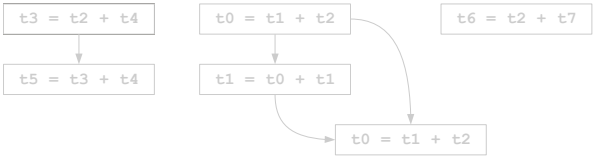


Instruction Scheduling



t3 = t2 + t4	t0 = t1 + t2
t5 = t3 + t4	t3 = t2 + t4
t0 = t1 + t2	t6 = t2 + t7
t1 = t0 + t1	t1 = t0 + t1
t0 = t1 + t2	t5 = t3 + t4
t6 = t2 + t7	

Instruction Scheduling



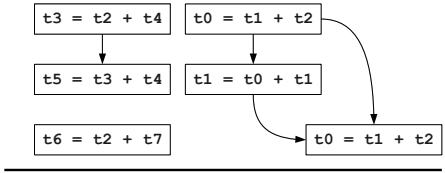

```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
t5 = t3 + t4
t0 = t1 + t2
```

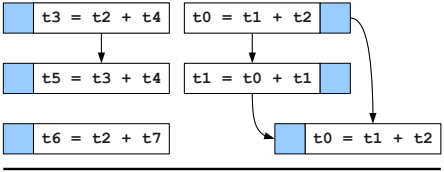
Problem

- Može postojati puno ispravnih topoloških uređenja grafa zavisnosti podataka
- Kako izabrati onaj redosled koji je dobar?
- U opštem slučaju, pronalaženje najboljeg rasporeda instrukcija je NP-težak problem.
- U praksi se koriste heuristike

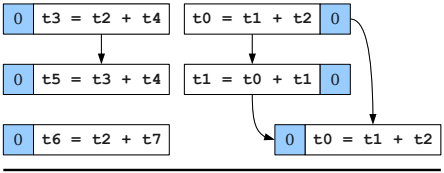
Instruction Scheduling



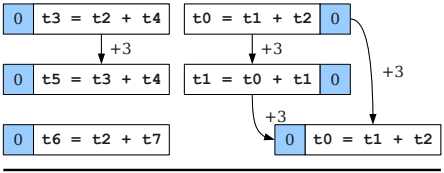
Instruction Scheduling



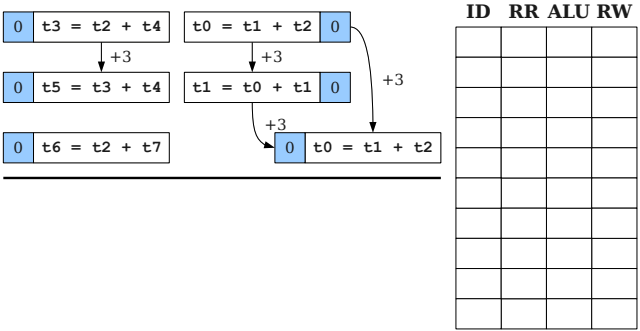
Instruction Scheduling



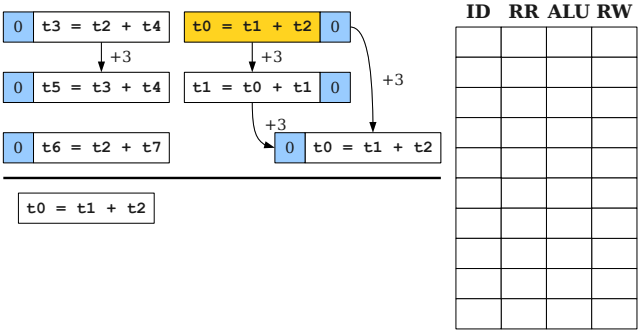
Instruction Scheduling



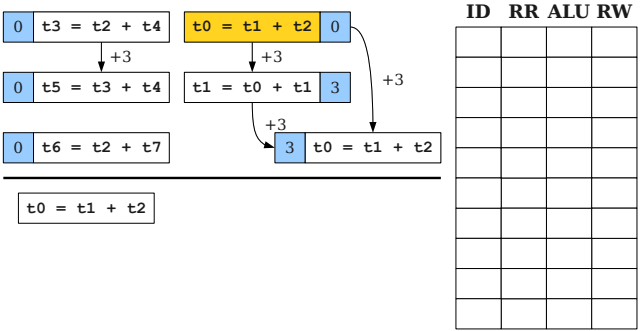
Instruction Scheduling



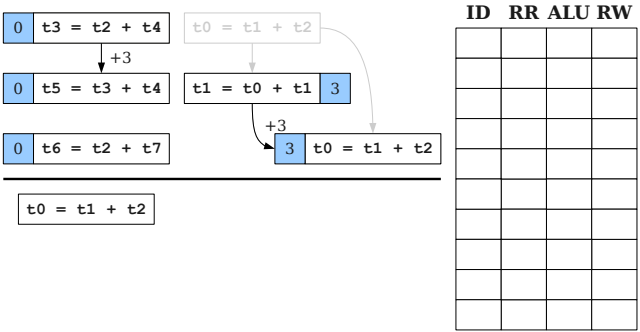
Instruction Scheduling



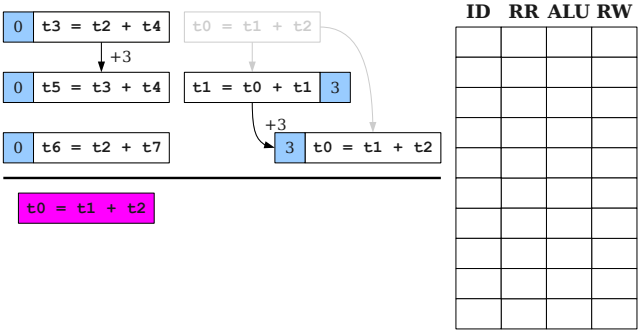
Instruction Scheduling



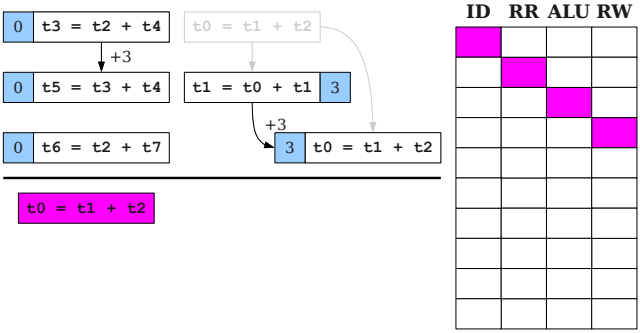
Instruction Scheduling



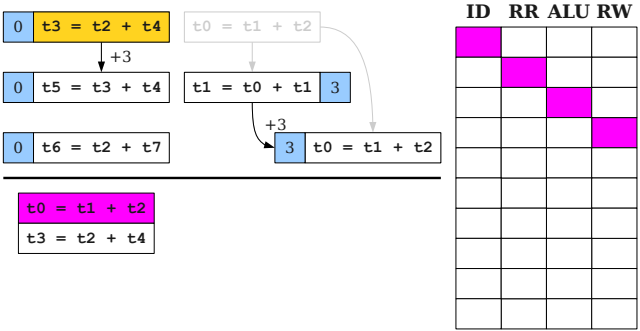
Instruction Scheduling



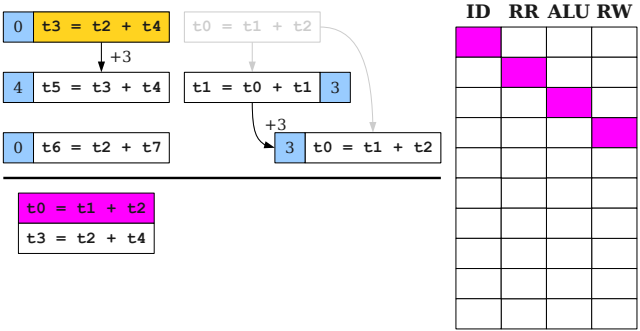
Instruction Scheduling



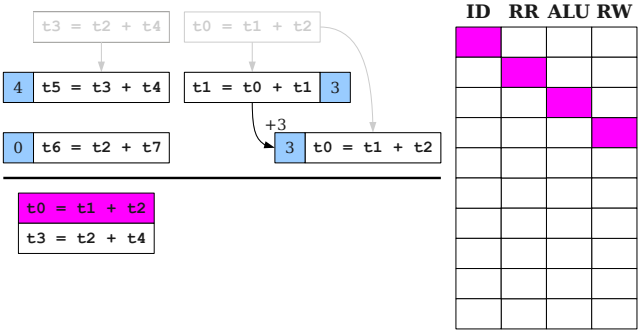
Instruction Scheduling



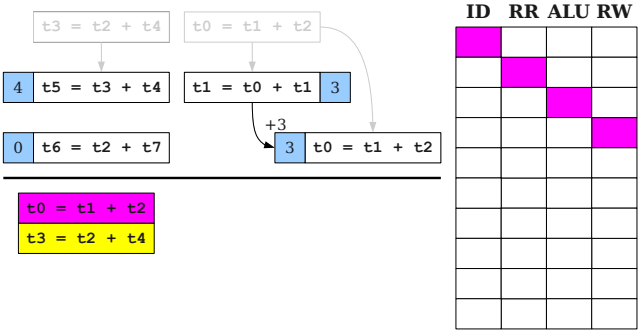
Instruction Scheduling



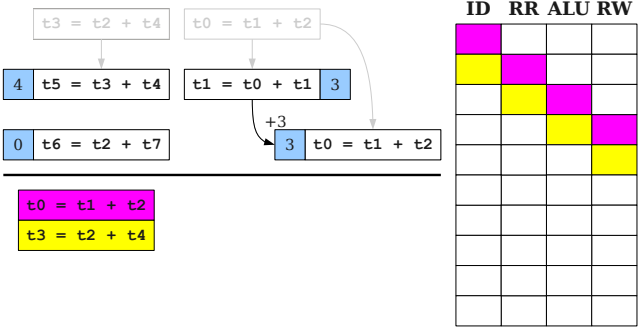
Instruction Scheduling



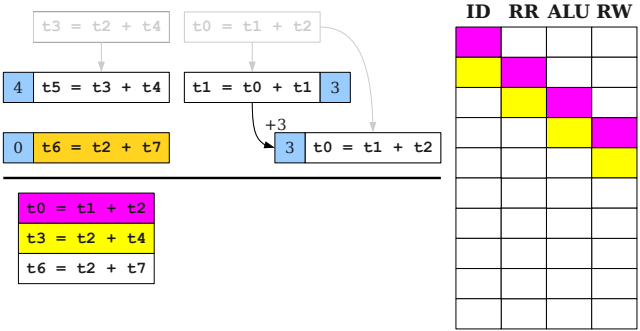
Instruction Scheduling



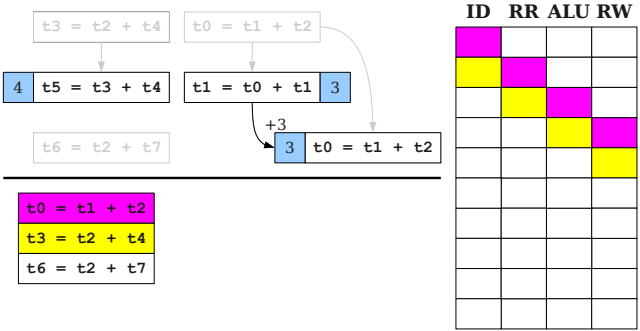
Instruction Scheduling



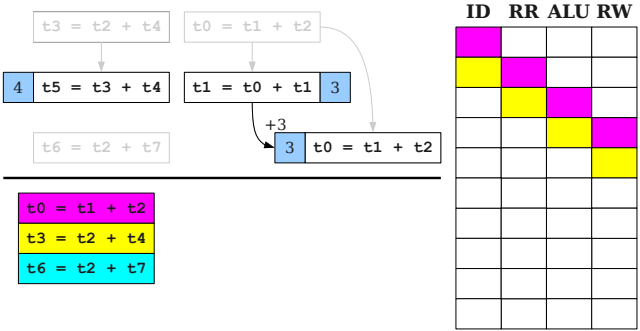
Instruction Scheduling



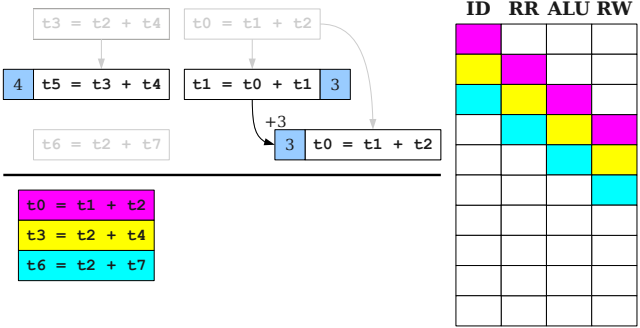
Instruction Scheduling



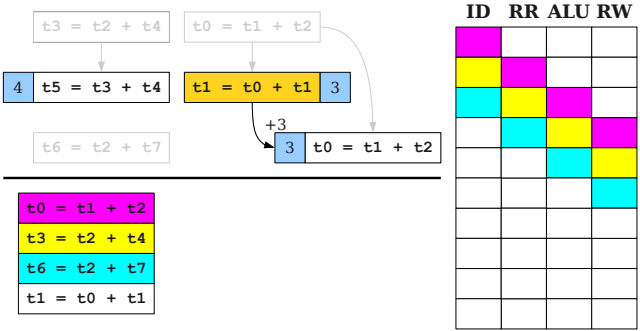
Instruction Scheduling



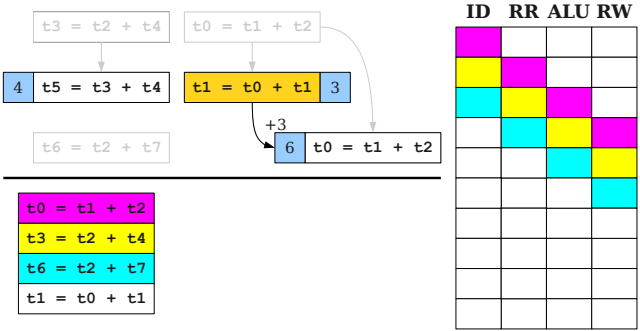
Instruction Scheduling



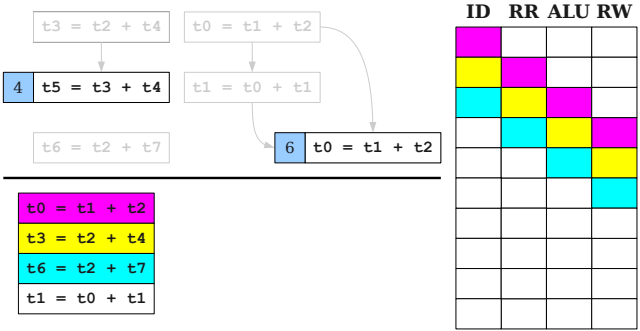
Instruction Scheduling



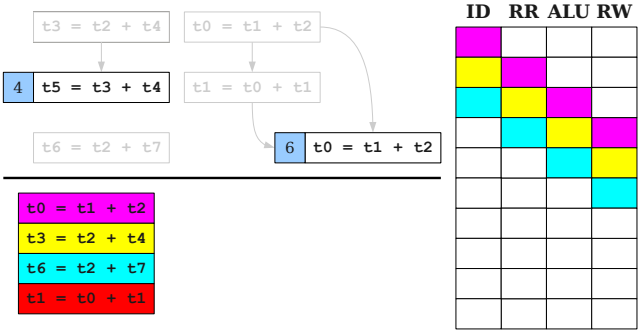
Instruction Scheduling



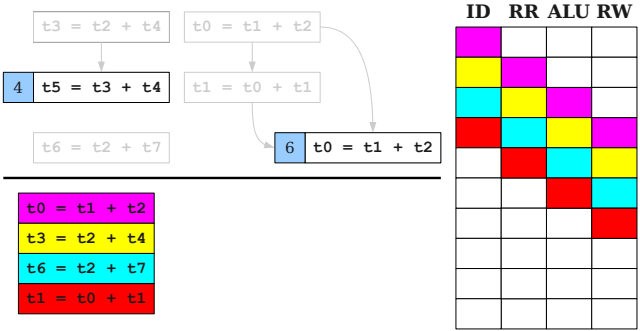
Instruction Scheduling



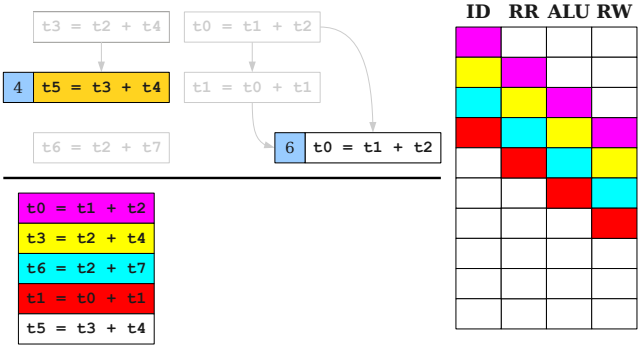
Instruction Scheduling



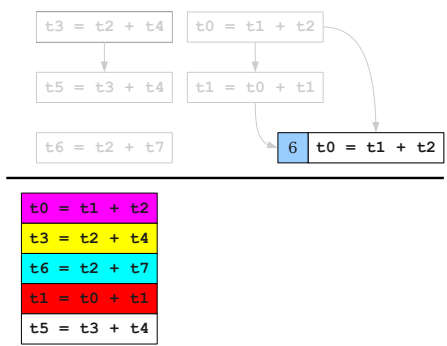
Instruction Scheduling



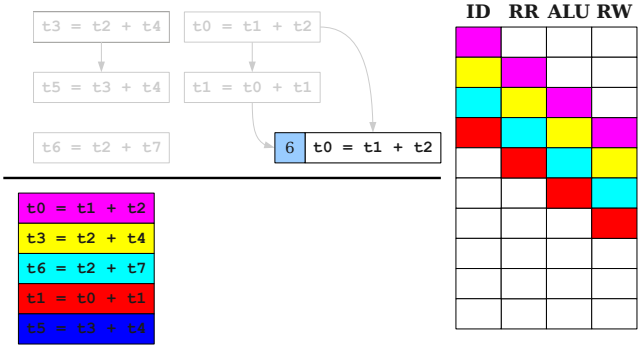
Instruction Scheduling



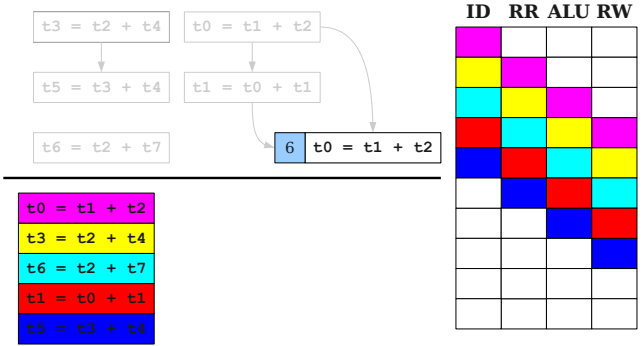
Instruction Scheduling



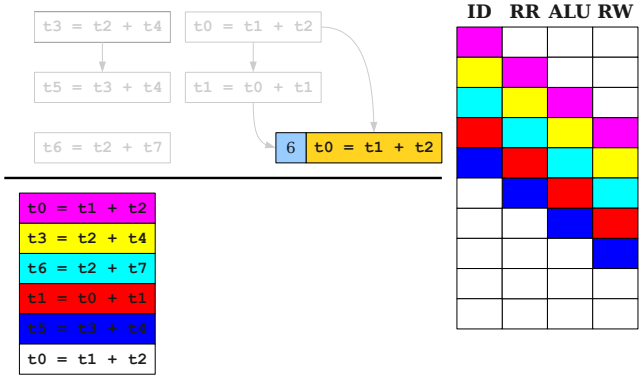
Instruction Scheduling



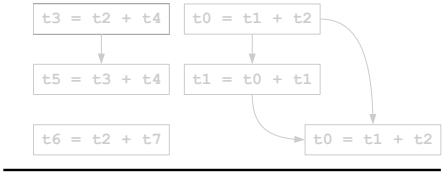
Instruction Scheduling

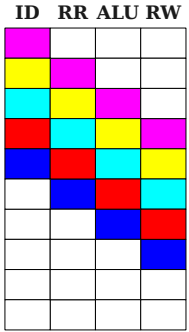


Instruction Scheduling

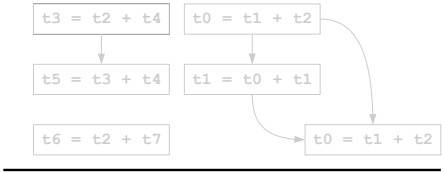


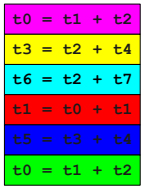
Instruction Scheduling

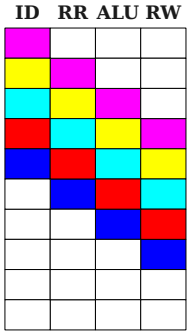




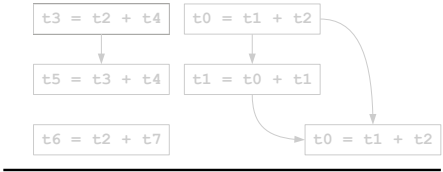
Instruction Scheduling

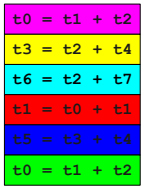


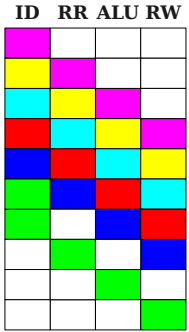




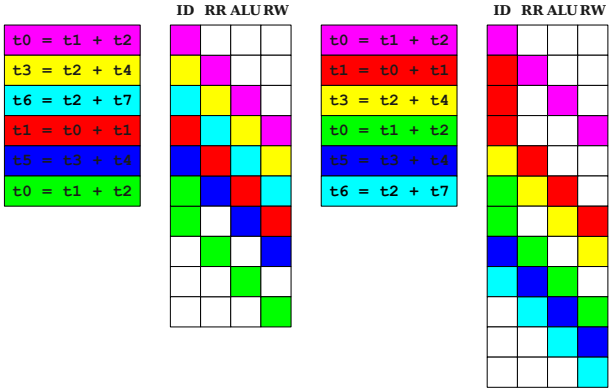
Instruction Scheduling







For Comparison



Raspoređivanje instrukcija

- Moderni kompajleri mogu da rade i značajno agresivnije raspoređivanje instrukcija sa ciljem da se dobiju bolje performanse programa
- Primeri ovih tehnika su razmotavanje petlji i *software pipelining*

Pregled

Upotreba keša

- Pored optimizacije raspoređivanje instrukcija, mogu se vršiti i optimizacije transformacije koda sa ciljem bolje upotrebe keša
- Upotreba keša se zasniva na dve vrste lokalnosti: vremenska i prostorna. Vremenska: ako je nekoj memoriji skoro pristupano, verovatno će joj biti ponovo pristupano uskoro. Prostorna: ako je nekoj memoriji skoro pristupano, verovatno će i njeni susedni objekti biti takođe uskoro korišćeni.
- Većina keš memorija je dizajnirano da iskoristi ove lokalnosti tako što se u kešu drže skoro adresirani objekti i tako što se u keš ubacuje i sadržaj memorije u blizini

Memory Caches

Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

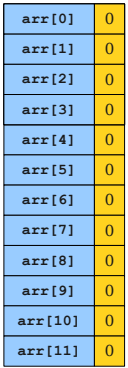
Memory Caches

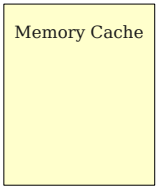
```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

arr[0]	0
arr[1]	0
arr[2]	0
arr[3]	0
arr[4]	0
arr[5]	0
arr[6]	0
arr[7]	0
arr[8]	0
arr[9]	0
arr[10]	0
arr[11]	0

Memory Caches

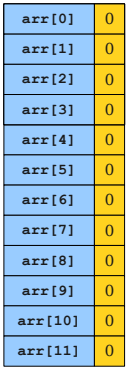
```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

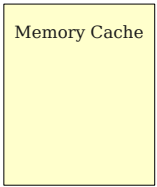




Memory Caches

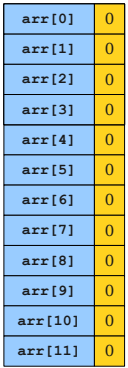
```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

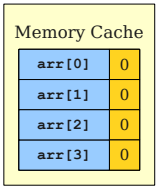




Memory Caches

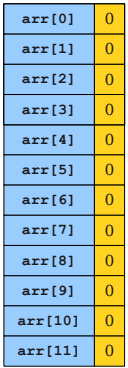
```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

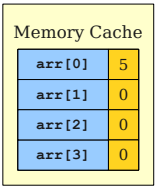




Memory Caches

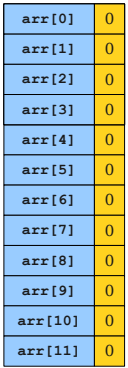
```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

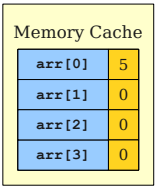




Memory Caches

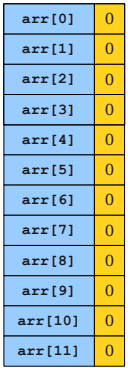
```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

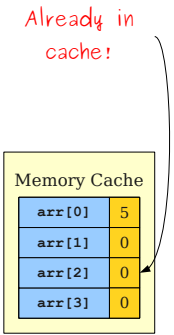




Memory Caches

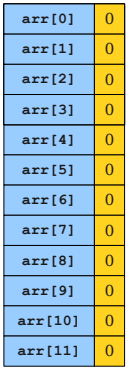
```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

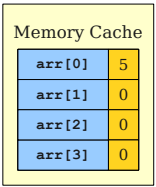




Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```





Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

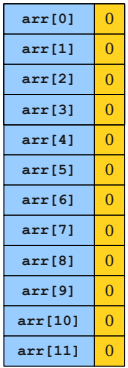
arr[0]	0
arr[1]	0
arr[2]	0
arr[3]	0
arr[4]	0
arr[5]	0
arr[6]	0
arr[7]	0
arr[8]	0
arr[9]	0
arr[10]	0
arr[11]	0

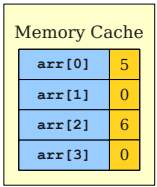
Memory Cache

arr[0]	5
arr[1]	0
arr[2]	6
arr[3]	0

Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```





Memory Caches

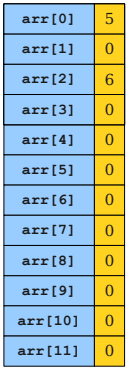
```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

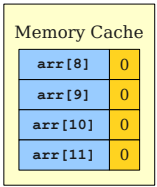
arr[0]	5
arr[1]	0
arr[2]	6
arr[3]	0
arr[4]	0
arr[5]	0
arr[6]	0
arr[7]	0
arr[8]	0
arr[9]	0
arr[10]	0
arr[11]	0

Memory Cache

Memory Caches

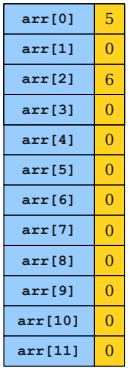
```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

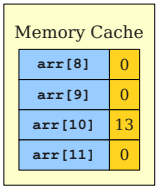




Memory Caches

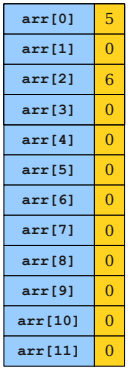
```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

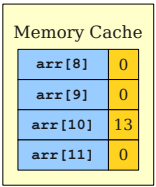




Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```





Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

arr[0]	5
arr[1]	0
arr[2]	6
arr[3]	0
arr[4]	0
arr[5]	0
arr[6]	0
arr[7]	0
arr[8]	0
arr[9]	0
arr[10]	13
arr[11]	0

Memory Cache

Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

arr[0]	5
arr[1]	0
arr[2]	6
arr[3]	0
arr[4]	0
arr[5]	0
arr[6]	0
arr[7]	0
arr[8]	0
arr[9]	0
arr[10]	13
arr[11]	0

Memory Cache

arr[0]	5
arr[1]	0
arr[2]	6
arr[3]	0

Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

arr[0]	5
arr[1]	0
arr[2]	6
arr[3]	0
arr[4]	0
arr[5]	0
arr[6]	0
arr[7]	0
arr[8]	0
arr[9]	0
arr[10]	13
arr[11]	0

Memory Cache

arr[0]	5
arr[1]	4
arr[2]	6
arr[3]	0

The Problem with Caches

The Problem with Caches

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

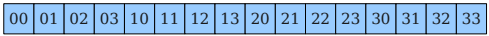
The Problem with Caches

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

The Problem with Caches

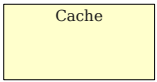
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```



The Problem with Caches

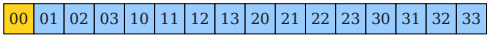
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

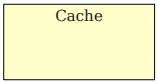
00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33



The Problem with Caches

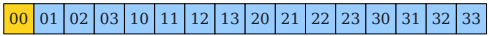
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

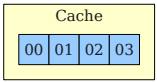




The Problem with Caches

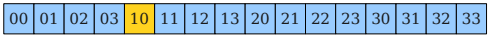
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

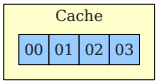




The Problem with Caches

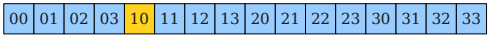
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

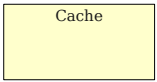




The Problem with Caches

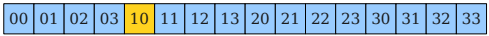
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

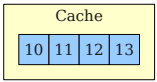




The Problem with Caches

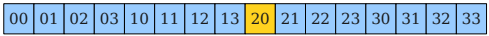
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

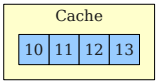




The Problem with Caches

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

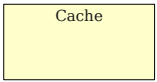




The Problem with Caches

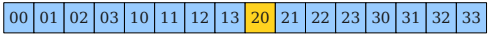
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

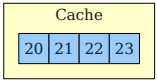
00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33



The Problem with Caches

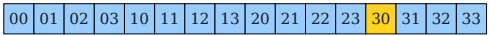
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

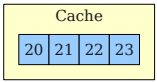




The Problem with Caches

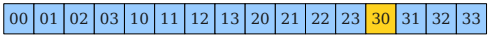
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

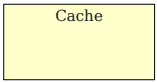




The Problem with Caches

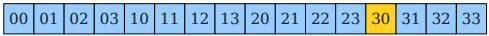
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

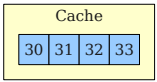




The Problem with Caches

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```





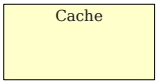
Poboljšanje lokalnosti

- Programeri obično pišu kod bez razumevanja o posledicama lokalnosti jer jezici ne prikazuju detalje memorije
- Neki kompajleri su sposobni da prepišu kod tako da se lokalnost iskoristi
- Primer takve optimizacije je preraspoređivanje petlji

Loop Reordering

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

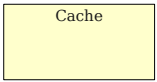
00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33



Loop Reordering

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

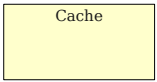
00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33



Loop Reordering

```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

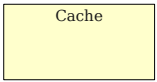
00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33



Loop Reordering

```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

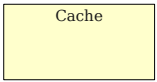
00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33



Loop Reordering

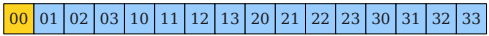
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

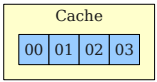
00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33



Loop Reordering

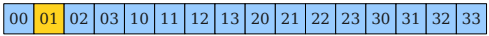
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

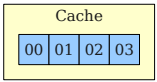




Loop Reordering

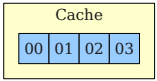
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```





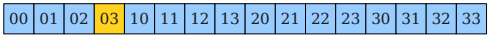
Loop Reordering

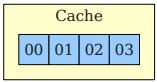
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```



Loop Reordering

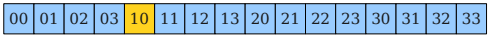
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

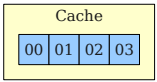




Loop Reordering

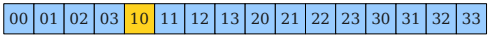
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

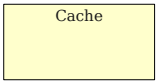




Loop Reordering

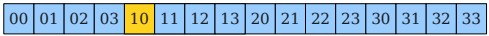
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

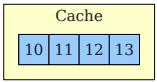




Loop Reordering

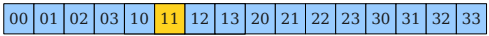
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

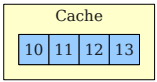




Loop Reordering

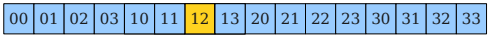
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

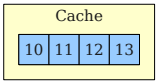




Loop Reordering

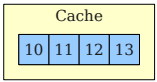
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```





Loop Reordering

```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```



Pregled

Optimizacije koda

- Postoje i razne druge optimizacije koje se mogu sprovesti na nivou izgenerisanog koda
- Posebno su bitne optimizacije koje se bave transformacijama petlji: optimizovanje koda koji se izvršava veliki broj puta je od suštinskog značaja

Razmotavanje petlje

```
int x;
for (x = 0; x < 100; x++)
{
    do_something(x);
}
```

```
int x;
 for (x = 0; x < 100; x += 5 )
 {
     do_something(x);
     do_something(x + 1);
     do_something(x + 2);
     do_something(x + 3);
     do_something(x + 4);
 }
```

- Šta se dobija?

Razmotavanje petlje

```
int x;
for (x = 0; x < 100; x++)
{
    do_something(x);
}
```

```
int x;
 for (x = 0; x < 100; x += 5 )
 {
     do_something(x);
     do_something(x + 1);
     do_something(x + 2);
     do_something(x + 3);
     do_something(x + 4);
 }
```

- Šta se dobija?
- Manji broj skokova, blokova i uslova — efikasniji kod
- Šta se gubi?

Razmotavanje petlje

```
int x;
for (x = 0; x < 100; x++)
{
    do_something(x);
}
```

```
int x;
 for (x = 0; x < 100; x += 5 )
 {
     do_something(x);
     do_something(x + 1);
     do_something(x + 2);
     do_something(x + 3);
     do_something(x + 4);
 }
```

- Šta se dobija?
- Manji broj skokova, blokova i uslova — efikasniji kod
- Šta se gubi?
- Duplikacija koda u petlji — veći kod
- Duplikacija koda van petlje (ako brojevi nisu deljivi)

Razmotavanje petlje

```
send(to, from, count)
register short *to, *from;
register count;
{
    register n = (count + 7) / 8;
    switch (count % 8) {
        case 0: *to = *from++;
        case 7: *to = *from++;
        case 6: *to = *from++;
        case 5: *to = *from++;
        case 4: *to = *from++;
        case 3: *to = *from++;
        case 2: *to = *from++;
        case 1: *to = *from++;
    }
    while (--n > 0) {
        *to = *from++;
        *to = *from++;
        *to = *from++;
        *to = *from++;
        *to = *from++;
        *to = *from++;
        *to = *from++;
        *to = *from++;
    }
}
```

Razmotavanje petlje — Duff's device

```
send(to, from, count)
register short *to, *from;
register count;
{
    register n = (count + 7) / 8;
    switch (count % 8) {
    case 0: do { *to = *from++;
    case 7:      *to = *from++;
    case 6:      *to = *from++;
    case 5:      *to = *from++;
    case 4:      *to = *from++;
    case 3:      *to = *from++;
    case 2:      *to = *from++;
    case 1:      *to = *from++;
            } while (--n > 0);
    }
}
```

Razdvajanje petlji — loop fission

```
int i, a[100], b[100];
for (i = 0; i < 100; i++) {
    a[i] = 1;
    b[i] = 2;
}
```

```
int i, a[100], b[100];
for (i = 0; i < 100; i++) {
    a[i] = 1;
}
for (i = 0; i < 100; i++) {
    b[i] = 2;
}
```

- Šta se dobija?

Razdvajanje petlji — loop fission

```
int i, a[100], b[100];
for (i = 0; i < 100; i++) {
    a[i] = 1;
    b[i] = 2;
}
```

```
int i, a[100], b[100];
for (i = 0; i < 100; i++) {
    a[i] = 1;
}
for (i = 0; i < 100; i++) {
    b[i] = 2;
}
```

- Šta se dobija?
- Bolja upotreba keša
- Mogućnost paralelizacije između procesora
- Šta se gubi?

Razdvajanje petlji — loop fission

```
int i, a[100], b[100];
for (i = 0; i < 100; i++) {
    a[i] = 1;
    b[i] = 2;
}
```

```
int i, a[100], b[100];
for (i = 0; i < 100; i++) {
    a[i] = 1;
}
for (i = 0; i < 100; i++) {
    b[i] = 2;
}
```

- Šta se dobija?
- Bolja upotreba keša
- Mogućnost paralelizacije između procesora
- Šta se gubi?
- Povećava se kod
- Dodaju se nove naredbe skoka

Spajanje petlji — loop fusion

- Obrnuti smer.
- Šta se dobija?

Spajanje petlji — loop fusion

- Obrnuti smer.
- Šta se dobija?
- Smanjuju se dodaci petlje (brojači, skokovi)
- Nezavisnost spojenih naredbi daje mogućnost paralelizma instrukcija
- Nekada se na ovaj način mogu smanjiti potrebe za čuvanjem međurezultata
- Šta se gubi?

Spajanje petlji — loop fusion

- Obrnuti smer.
- Šta se dobija?
- Smanjuju se dodaci petlje (brojači, skokovi)
- Nezavisnost spojenih naredbi daje mogućnost paralelizma instrukcija
- Nekada se na ovaj način mogu smanjiti potrebe za čuvanjem međurezultata
- Šta se gubi?
- Veći pritisak na registre
- Lošija upotreba keša

Razdvajanje prve/poslednej iteracije — Loop peeling

- Loop peeling je specijani slučaj od loop splitting. It splits any problematic first (or last) few iterations from the loop and performs them outside of the loop body. (gcc 3.4)
- Loop splitting is a compiler optimization technique. It attempts to simplify a loop or eliminate dependencies by breaking it into multiple loops which have the same bodies but iterate over different contiguous portions of the index range.

```
int p = 10;
for (int i=0; i<10; ++i)
{
  y[i] = x[i] + x[p];
  p = i;
}
```

```
y[0] = x[0] + x[10];
for (int i=1; i<10; ++i)
{
  y[i] = x[i] + x[i-1];
}
```

Software pipelining

- Iskoristiti na najbolji način raspoređivanje instrukcija u petljama
- Na primer, ukoliko u telu petlje imamo nekoliko instrukcija koje zavise međusobno, ali ne zavise od naredne iteracije, onda to treba iskoristiti: razmotavanje petlje + raspoređivanje instrukcija

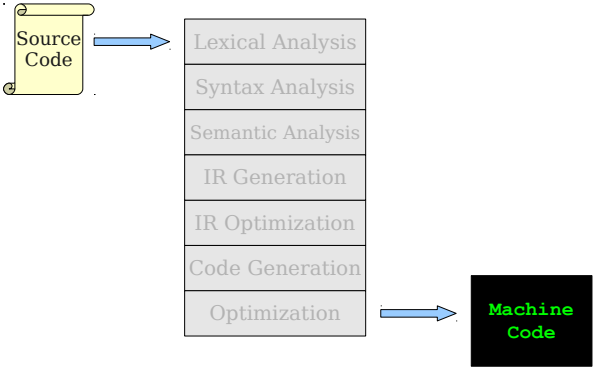
```
for i = 1 to bignumber
  A(i)  //load
  B(i)  //add
  C(i)  //store
end
```

```
for i = 1 to (bignumber/3) step 3
  A(i)
  A(i+1)
  A(i+2)
  B(i)
  B(i+1)
  B(i+2)
  C(i)
  C(i+1)
  C(i+2)
end
```

Optimizacije petlji

- Svaka od ovih optimizacija zahteva kompleksnu cost/benefit analizu
- U zavisnosti od ostalih parametara, donosi se odluka koja optimizacija se kada koristi

Where We Are


Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Machine Code

Pregled

Literatura

- (The Dragon Book) Compilers: Principles, Techniques, and Tools Alfred V. Aho,Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman
- Kompajleri Stanford
  `https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/`