اصول طراحی کامپایلر

درس ۲۲

بهینه‌سازی کد میانی

Intermediate Code Optimization

کاظم فولادی قلعه

دانشکده مهندسی، پردیس فارابی

دانشگاه تهران

http://courses.fouladi.ir/compiler

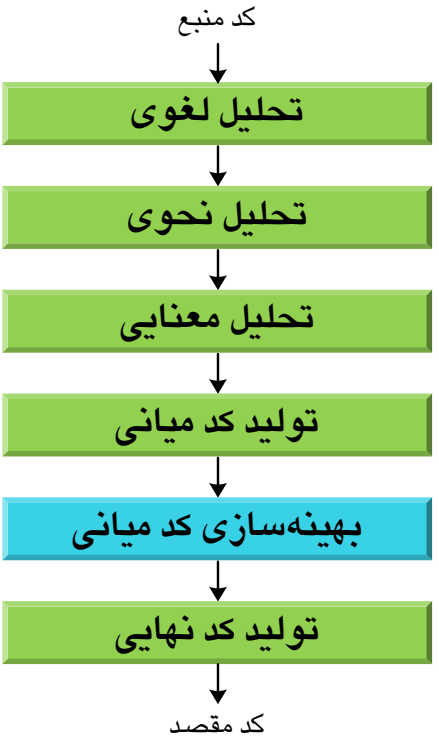
بهینه‌سازی کد میانی

۱

مقدمه

مراحل فرآیند کامپایل در یک نگاه

بهینه‌سازی کد میانی



بهینه‌سازی کد میانی

بهینه‌سازی کد میانی

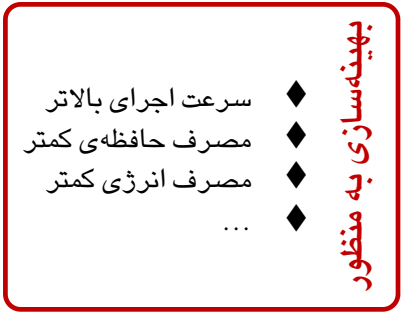
مثال


```
temp1 := inttofloat(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

```
temp1 := id3 * 60.0
id1 := id2 + temp1
```

بهینه‌سازی کد میانی

اهداف



بهینه‌سازی تضمین نمی‌کند که کد حاصل بهترین کد ممکن باشد!

بهینه‌سازی کد میانی

انواع بهینه‌سازی

بهینه‌سازی‌های مستقل از ماشین
Machine-Independent Optimization

هیچ ملاحظه‌ای در مورد
خصوصیات ماشین مقصد وارد نمی‌شود.

بهینه‌سازی‌های وابسته به ماشین
Machine-Dependent Optimization

خصوصیات ماشین مقصد برای بهینه‌سایز کد
استفاده می‌شود.

بهینه‌سازی کد میانی

تکنیک‌های بهینه‌سازی

تحلیل جریان کنترل *Control-Flow Analysis*	تحلیل جریان داده‌ها *Data-Flow Analysis*
شناسایی حلقه‌ها در گراف جریان برنامه (حلقه‌ها معمولاً گزینه‌های خوبی برای بهبود هستند.)	جمع‌آوری اطلاعات در مورد چگونگی استفاده از متغیرها در برنامه

تکنیک‌های بهبود کد معمولاً ترکیبی از تحلیل‌های جریان کنترل و تحلیل‌های جریان داده هستند.

بهینه‌سازی کد میانی

ضوابط تبدیل‌های بهبود کد

یک تبدیل باید معنای برنامه را حفظ کند.

یک تبدیل باید سرعت اجرای برنامه در حالت متوسط را به میزانی قابل اندازه‌گیری افزایش دهد.

بهترین تبدیل‌ها آنهایی هستند که حداکثر منفعت را با حداقل تلاش به‌دست می‌دهند.

از بهینه‌سازی کد برای برنامه‌هایی که به‌ندرت اجرا می‌شوند باید اجتناب شود.

از بهینه‌سازی کد هنگام اشکال‌زدایی برنامه باید اجتناب شود.

تذکر: بهبودهای اساسی با بهبود کد منبع برنامه توسط برنامه‌نویس به‌وجود می‌آید.
برنامه‌نویس همیشه مسئول یافتن بهترین ساختمان‌داده‌ها و الگوریتم‌ها برای حل یک مسئله است.

۲

تبدیل‌های نمونه برای بهینه‌سازی کد میانی

گراف جریان برنامه

در طول تحلیل جریان کنترل، یک برنامه به‌صورت گراف جریان نمایش داده می‌شود.

گراف جریان *Flow Graph*	
گره‌ها *Nodes*	**یال‌ها** *Edges*
بلاک‌های پایه‌ی برنامه	نمایش جریان کنترل برنامه

گراف جریان برنامه

بلاک پایه

بلاک پایه
Basic Block

دنباله‌ای از دستورالعمل‌های پی‌درپی که جریان کنترل از ابتدای آن آغاز می‌شود و تا انتهای آن بدون پرش یا توقف خاتمه می‌یابد.

غیر از اولین دستورالعمل بلاک پایه، هیچ مقصد پرش دیگری داخل بلاک پایه وجود ندارد.

یک برنامه‌ی نمونه برای بهینه‌سازی

مرتب‌سازی سریع (QuickSort)

```
void quicksort(int m, int n)
   /* recursively sorts a[m] through a[n] */
{
   int i, j;
   int v, x;
   if (n <= m) return;
   /* fragment begins here */
   i = m-1; j = n; v = a[n];
   while (1)
   {
      do i = i+1; while (a[i] < v);
      do j = j-1; while (a[j] > v);
      if (i >= j) break;
      x = a[i]; a[i] = a[j]; a[j] = x; /* swap a[i], a[j] */
   }
   x = a[i]; a[i] = a[n]; a[n] = x; /* swap a[i], a[n] */
   /* fragment ends here */
   quicksort(m,j); quicksort(i+1,n);
}
```

برای نمایش تأثیر تکنیک‌های مختلف بهینه‌سازی از قطعه کد این برنامه‌ی زبان C استفاده می‌کنیم.

یک برنامه‌ی نمونه برای بهینه‌سازی

مرتب‌سازی سریع (QuickSort): کد میانی سه‌آدرسی

```
(1)     i = m-1
(2)     j = n
(3)     t1 = 4*n
(4)     v = a[t1]
(5)     i = i+1
(6)     t2 = 4*i
(7)     t3 = a[t2]
(8)     if t3<v goto (5)
(9)     j = j-1
(10)    t4 = 4*j
(11)    t5 = a[t4]
(12)    if t5>v goto (9)
(13)    if i>=j goto (23)
(14)    t6 = 4*i
(15)    x = a[t6]
(16)    t7 = 4*i
(17)    t8 = 4*j
(18)    t9 = a[t8]
(19)    a[t7] = t9
(20)    t10 = 4*j
(21)    a[t10] = x
(22)    goto (5)
(23)    t11 = 4*i
(24)    x = a[t11]
(25)    t12 = 4*i
(26)    t13 = 4*n
(27)    t14 = a[t13]
(28)    a[t12] = t14
(29)    t15 = 4*n
(30)    a[t15] = x
```

یک برنامه‌ی نمونه برای بهینه‌سازی

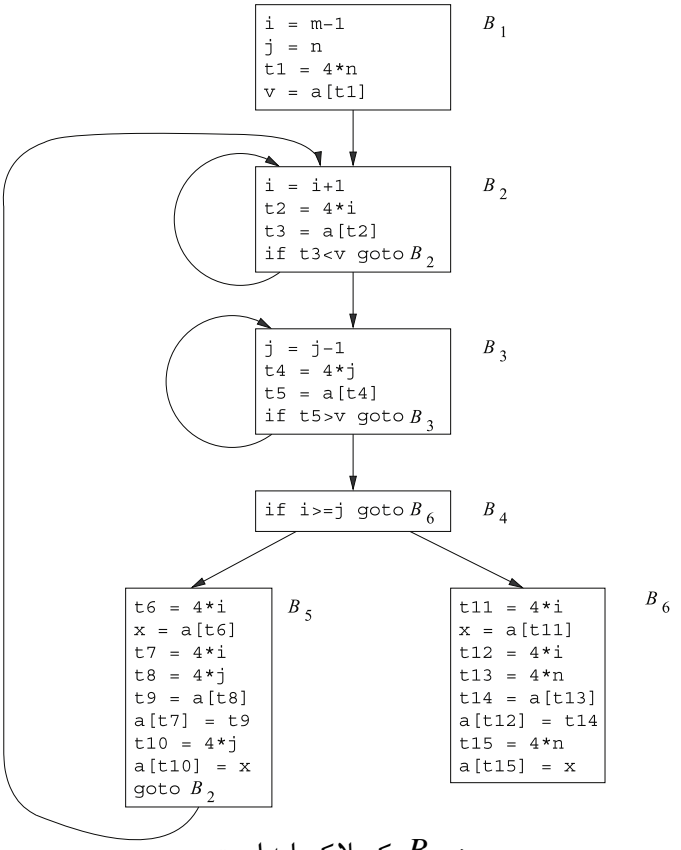
مرتب‌سازی سریع (QuickSort): کد میانی سه‌آدرسی: تعیین مرز بلاک‌های پایه

```
(1)     i = m-1
(2)     j = n
(3)     t1 = 4*n
(4)     v = a[t1]
(5)     i = i+1
(6)     t2 = 4*i
(7)     t3 = a[t2]
(8)     if t3<v goto (5)
(9)     j = j-1
(10)    t4 = 4*j
(11)    t5 = a[t4]
(12)    if t5>v goto (9)
(13)    if i>=j goto (23)
(14)    t6 = 4*i
(15)    x = a[t6]
(16)    t7 = 4*i
(17)    t8 = 4*j
(18)    t9 = a[t8]
(19)    a[t7] = t9
(20)    t10 = 4*j
(21)    a[t10] = x
(22)    goto (5)
(23)    t11 = 4*i
(24)    x = a[t11]
(25)    t12 = 4*i
(26)    t13 = 4*n
(27)    t14 = a[t13]
(28)    a[t12] = t14
(29)    t15 = 4*n
(30)    a[t15] = x
```

مرزها: ۱) نقاط پرش ۲) نقاط مقصد پرش

یک برنامه‌ی نمونه برای بهینه‌سازی

مرتب‌سازی سریع (QuickSort): گراف جریان



هر B_i یک بلاک پایه است.

گستره‌ی بهینه‌سازی

گستره‌ی بهینه‌سازی	
بهینه‌سازی محلی *Local Optimization*	**بهینه‌سازی سراسری** *Global Optimization*
تنها با دستورهای موجود در یگ بلاک پایه سروکار دارد.	با کل برنامه سروکار دارد.

منابع اصلی بهینه‌سازی

تبدیل‌های قابل اعمال روی بلاک پایه

هر بلاک پایه، مجموعه‌ای از عبارت‌ها را محاسبه می‌کند.

می‌توان تبدیل‌هایی را بر روی بلاک پایه اعمال کرد،
بدون اینکه عبارت محاسبه شده توسط آن بلاک تغییر کند

تبدیل‌های قابل اعمال روی بلاک پایه
حذف زیرعبارت مشترک *Common Subexpression Elimination*
انتشار کپی *Copy Propagation*
حذف کد مرده *Dead-Code Elimination*
جادادن ثابت *Constant Folding*

تبدیل‌های قابل اعمال روی بلاک پایه

حذف زیرعبارت مشترک

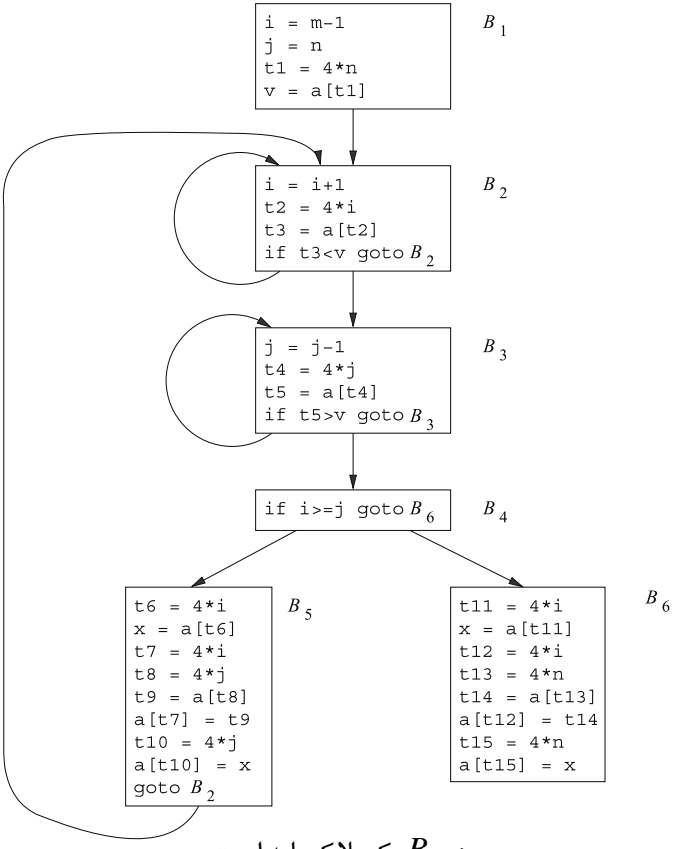
زیرعبارت مشترک
Common Subexpression

یک وقوع از عبارت E یک **زیرعبارت مشترک** نامیده می‌شود
اگر (۱) E قبلاً محاسبه شده باشد و
(۲) مقادیر متغیرهای موجود در E از محاسبه‌ی قبلی تغییر نکرده باشد.

قاعده‌ی انتشار کپی:
زیرعبارت مشترک را فقط در نخستین بار محاسبه می‌کنیم و در سایر دفعات از مقدار آن استفاده می‌کنیم.

یک برنامه‌ی نمونه برای بهینه‌سازی

مرتب‌سازی سریع (QuickSort): گراف جریان



هر B_i یک بلاک پایه است.

تبدیل‌های قابل اعمال روی بلاک پایه

حذف زیرعبارت مشترک: بهینه‌سازی محلی: مثال

قبل از حذف زیرعبارت مشترک

```
t6 = 4*i
x = a[t6]
t7 = 4*i
t8 = 4*j
t9 = a[t8]
a[t7] = t9
t10 = 4*j
a[t10] = x
goto B2
```

B_5

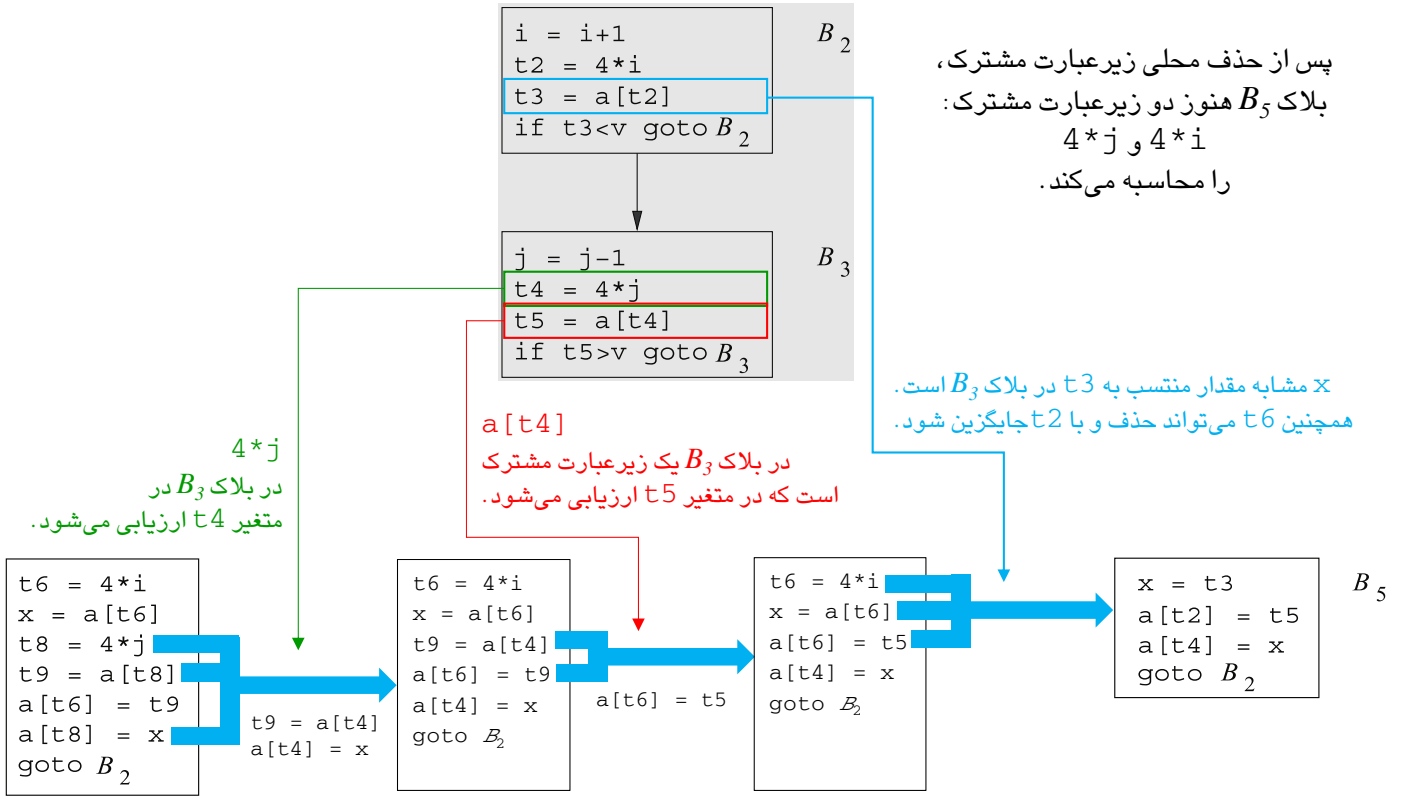
بعد از حذف زیرعبارت مشترک

```
t6 = 4*i
x = a[t6]
t8 = 4*j
t9 = a[t8]
a[t6] = t9
a[t8] = x
goto B2
```

B_5

تبدیل‌های قابل اعمال روی بلاک پایه

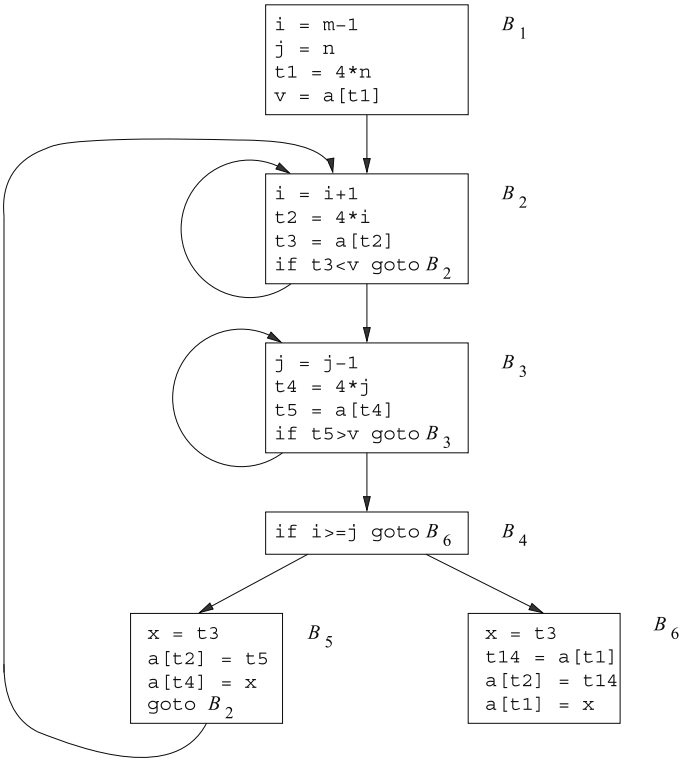
حذف زیرعبارت مشترک: بهینه‌سازی سراسری: مثال



تبدیل‌های قابل اعمال روی بلاک پایه

حذف زیرعبارت مشترک: گراف جریان حاصل

پس از حذف زیرعبارت‌های مشترک محلی و سراسری دربلاک‌های B_5 و B_6



تبدیل‌های قابل اعمال روی بلاک پایه

تعیین زیرعبارت مشترک

استفاده از گراف‌های جهت‌دار بدون دور (DAG) برای تعیین زیرعبارت‌های مشترک یک بلاک پایه

این گراف چگونگی استفاده‌ی مجدد از زیرعبارت‌ها در یک بلاک را نشان می‌دهد.

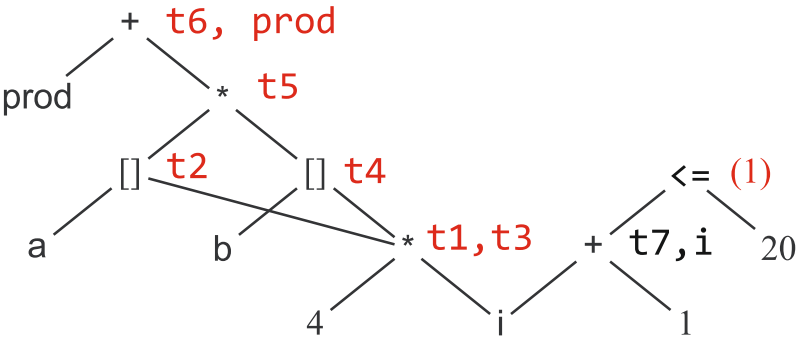
گراف‌های جهت‌دار بدون دور (DAG) برای یک بلاک پایه	
گره‌ها *Nodes*	**یال‌ها** *Edges*
❋ گره‌های داخلی: حاوی نماد عملگر ❋ گره‌های برگ: حاوی شناسه‌های یکتا (نام متغیرها و ثابت‌ها) هر گره می‌تواند دارای لیستی از متغیرهای وابسته باشد تا نشان دهد مقدار آن متغیرها در آن گره محاسبه می‌شود.	سلسله‌مراتب محاسبه

تبدیل‌های قابل اعمال روی بلاک پایه

گراف‌های جهت‌دار بدون دور (DAG) برای یک بلاک پایه: مثال

بلاک پایه

```
(1)     t1 := 4 * i
(2)     t2 := a[t1]
(3)     t3 := 4 * i
(4)     t4 := b[t3]
(5)     t5 := t2 * t4
(6)     t6 := prod + t5
(7)     prod := t6
(8)     t7 := i + 1
(9)     i := t7
(10)    if i <= 20 goto (1)
```



گراف‌های جهت‌دار بدون دور (DAG) برای تعیین زیرعبارت‌های مشترک بلاک پایه

تبدیل‌های قابل اعمال روی بلاک پایه

انتشار کپی

دستور کپی
Copy Instruction

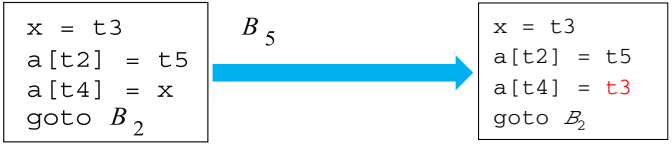
دستور y =: x (مقدار متغیر y در متغیر x کپی می‌شود)

قاعده‌ی انتشار کپی:
با داشتن دستور کپی $y =: x$ پس از این دستور هر جا ممکن بود به‌جای x از y استفاده می‌کنیم.

تبدیل‌های قابل اعمال روی بلاک پایه

انتشار کپی: مثال

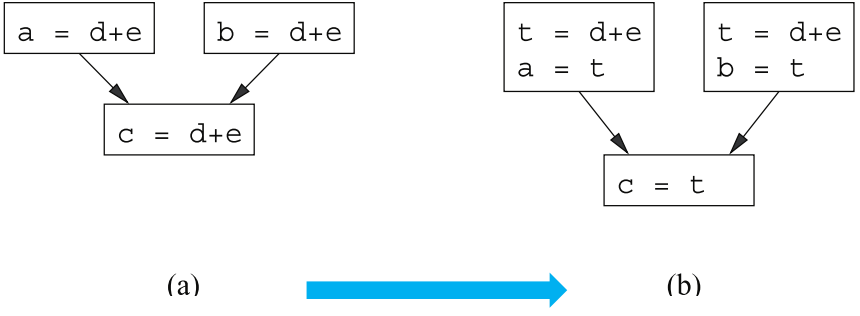
اعمال قاعده‌ی انتشار کپی روی بلاک B_5



این تبدیل به‌همراه حذف کد کرده، در مجموع امکان حذف انتساب x := t3 را فراهم می‌کند.

تبدیل‌های قابل اعمال روی بلاک پایه

انتشار کپی: مثال



نمونه‌ی کپی‌های ایجاد شده پس از حذف زیرعبارت مشترک
و سپس استفاده از قاعده‌ی انتشار کپی

تبدیل‌های قابل اعمال روی بلاک پایه

حذف کد مرده

متغیر زنده *Live Variable*	یک متغیر در یک نقطه از برنامه **زنده** است، اگر مقدار آن بتواند متعاقباً استفاده شود.
متغیر مرده *Dead Variable*	یک متغیر در یک نقطه از برنامه **مرده** است، اگر مقدار آن متعاقباً استفاده نشود.
کد مرده *Dead Code*	بخشی از یک کد مرده است، اگر داده‌های محاسبه شده در آن در هیچ جای دیگری استفاده نشود.

قاعده‌ی حذف کد مرده

کد مرده از برنامه کنار گذاشته می‌شود.

کد مرده ممکن است حاصل تبدیل انتشار کپی باشد ⇐

حذف کد مرده معمولاً به‌همراه تبدیل انتشار کپی استفاده می‌شود.

تبدیل‌های قابل اعمال روی بلاک پایه

حذف کد مرده: مثال

x در هیچ جای دیگری از برنامه بعد از بلاک B_5 استفاده نمی‌شود
پس x یک متغیر مرده و کد انتساب آن، کد مرده است و حذف می‌شود.

تبدیل‌های قابل اعمال روی بلاک پایه

جا دادن ثابت

جادادن ثابت
Constant Folding

تبدیلی است که یک عبارت را با یک مقدار ثابت جایگزین می‌کند.

ممکن است بتوانیم در زمان کامپایل متوجه شویم که مقدار یک عبارت (یا متغیر) ثابت است.

قاعده‌ی جادادن ثابت

هرگاه در زمان کامپایل متوجه شدیم که مقدار یک عبارت ثابت است، محاسبه‌ی آن را با آن مقدار ثابت جایگزین می‌کنیم.

جادادن ثابت برای کشف کد مرده مفید است.

تبدیل‌های قابل اعمال روی بلاک پایه

جا دادن ثابت‌ها: مثال

```
if x goto L
```

اگر با جادادن ثابت متوجه شویم که در دستور شرطی فوق، x همیشه نادرست است،
می‌توانیم آزمون شرط if و پرش به برچسب L را حذف کنیم.

تبدیل‌های قابل اعمال روی بلاک پایه

بهینه‌سازی حلقه‌ها

از آنجا که بدنه‌ی حلقه‌ها معمولاً چندین بار اجرا می‌شوند،
اگر دستورهای موجود در حلقه‌ها را بهینه کنیم، در زمان اجرای برنامه بهبود قابل ملاحظه‌ای اتفاق می‌افتد.

تکنیک‌های بهینه‌سازی حلقه‌ها		
جابجایی کد *Code Motion*	**کاهش دشواری** *Reduction in Strength*	**حذف متغیر استقرایی** *Induction-Variable Elimination*

تبدیل‌های قابل اعمال روی بلاک پایه

بهینه‌سازی حلقه‌ها: تکنیک جابجایی کد

قاعده‌ی جابجایی کد

اگر محاسبه‌ی یک عبارت نامتغیر در حلقه *(loop-invariant)* باشد، کد این محاسبه، قبل از حلقه قرار می‌گیرد.

تبدیل‌های قابل اعمال روی بلاک پایه

بهینه‌سازی حلقه‌ها: تکنیک جابجایی کد: مثال

تبدیل‌های قابل اعمال روی بلاک پایه

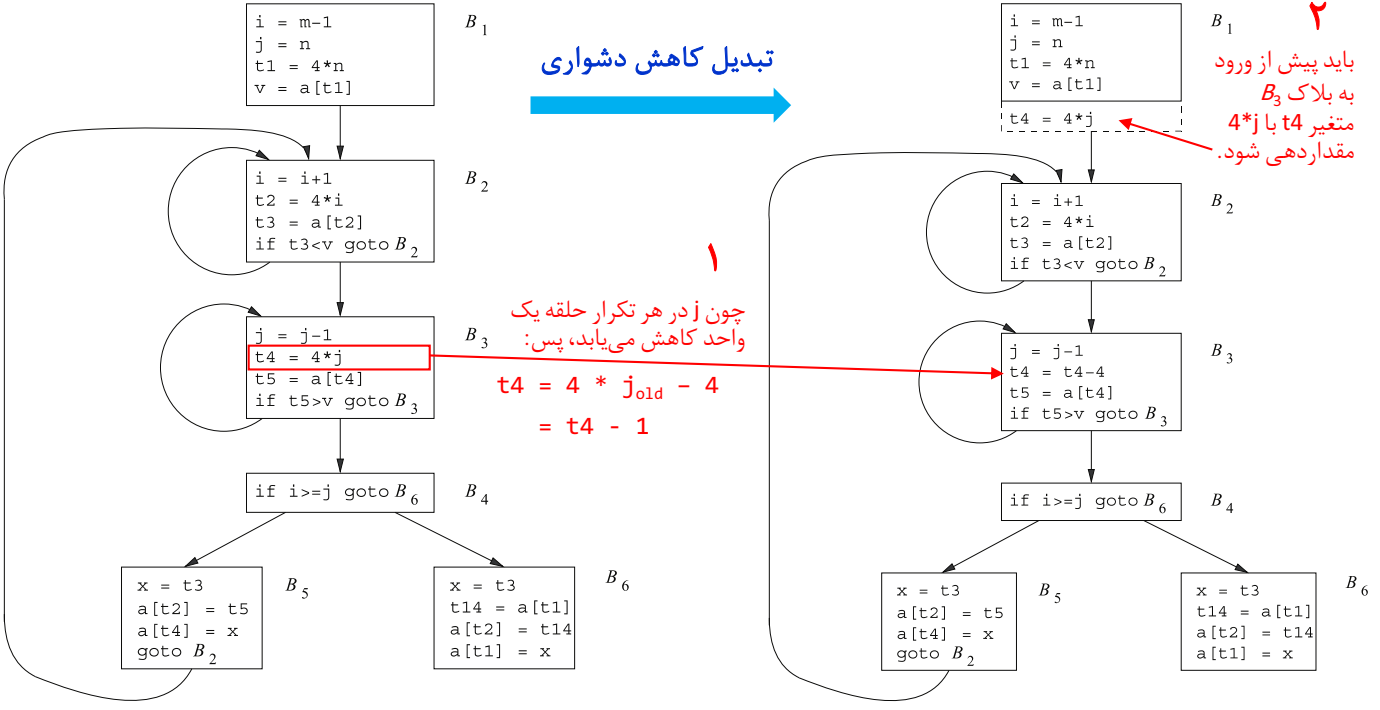
بهینه‌سازی حلقه‌ها: تکنیک کاهش دشواری

قاعده‌ی کاهش دشواری

منظور از کاهش دشواری، جایگزینی یک محاسبه با یک محاسبه‌ی ارزان‌تر است.

تبدیل‌های قابل اعمال روی بلاک پایه

بهینه‌سازی حلقه‌ها: تکنیک کاهش دشواری: مثال



نتیجه: جایگزینی یک ضرب با یک تفریق در داخل حلقه، موجب افزایش سرعت اجرای کد می‌شود.

تبدیل‌های قابل اعمال روی بلاک پایه

بهینه‌سازی حلقه‌ها: تکنیک حذف متغیرهای استقرایی

متغیر استقرایی
Induction Variable

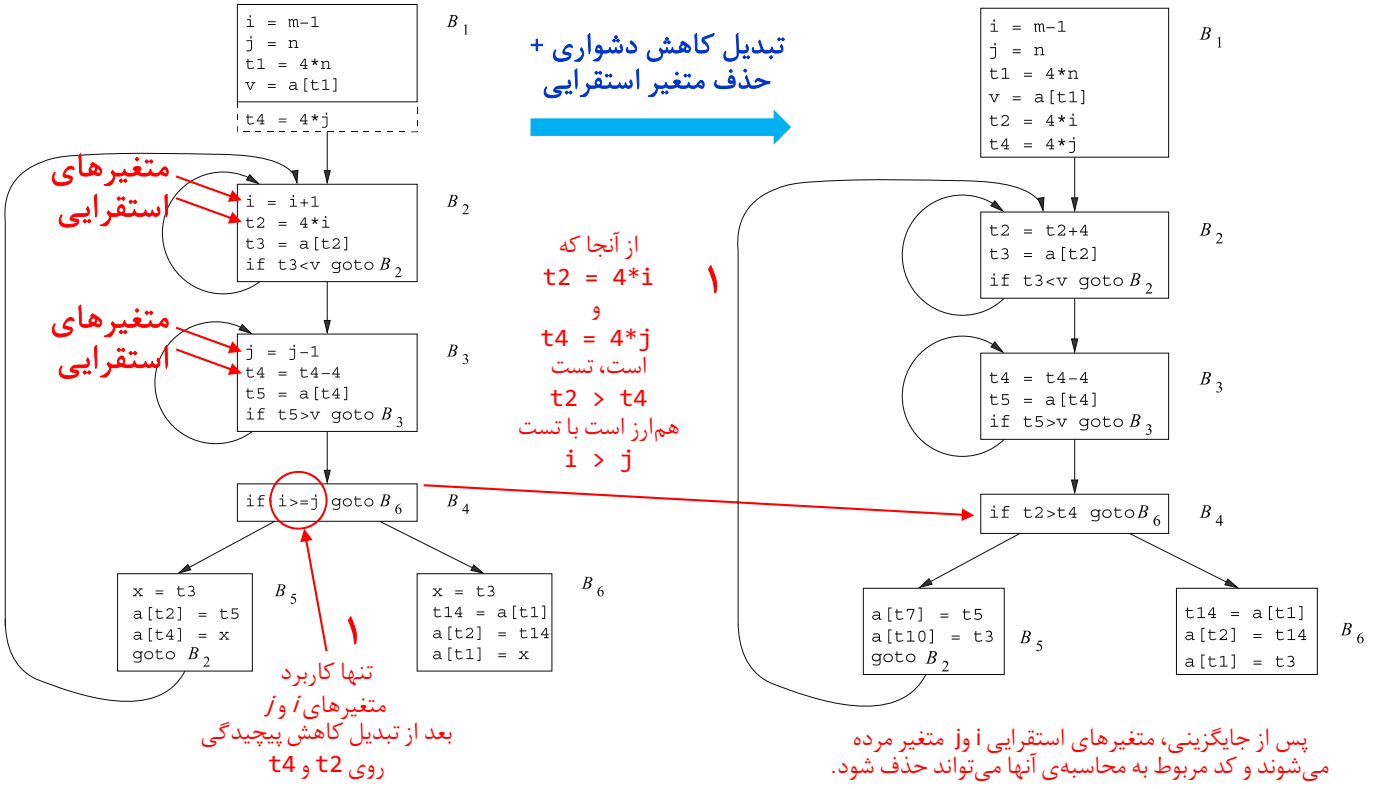
متغیر x یک متغیر استقرایی یک حلقه است اگر
هر بار تغییر مقدار x به صورت **افزایش** (یا کاهش) به یک میزان **ثابت** باشد.

قاعده‌ی حذف متغیر استقرایی
اگر در یک حلقه چند متغیر استقرایی داشته باشیم، یکی را نگه می‌داریم و بقیه را بر حسب آن بازنویسی می‌کنیم.

یک نمونه‌ی متداول این است که:
یک متغیر استقرایی مانند i یک آرایه را اندیس‌گذاری می‌کند و
یک متغیر استقرایی دیگر مانند t آفست واقعی برای دسترسی به عناصر آرایه باشد.

تبدیل‌های قابل اعمال روی بلاک پایه

بهینه‌سازی حلقه‌ها: تکنیک حذف متغیرهای استقرایی: مثال

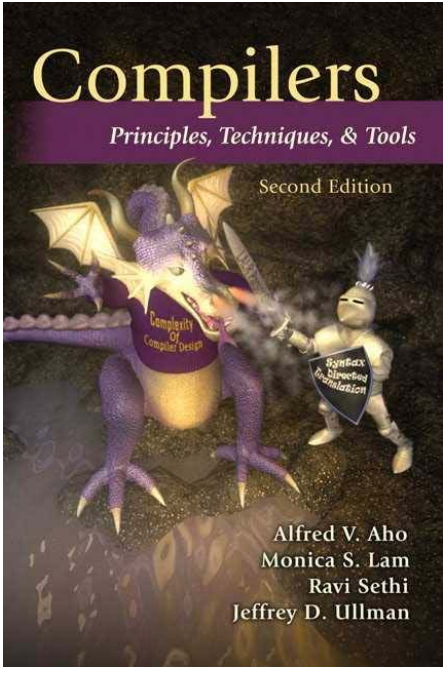


بهینه‌سازی کد میانی

۳

منابع

منبع اصلی



A. V. Aho, M. S. Lam, R. Sethi, J. D. Ullman,
Compilers: Principles, Techniques and Tools,
Second Edition, Addison-Wesley, 2007.

Chapter 9 (9.1)