ארגון המחשב ומערכות הפעלה

אביב תשפ"ד

תרגול 8 - CPE

CPE

- clock cycles per element -CPE

זהו מושג באופטימיזציה של תכניות

דרך נוחה למדידת ביצועים במקרה של וקטור / רשימת נתונים

מה זה אומר?

למשל, בתכנית בה השלבים מתבצעים בהדרגתיות, ובנוסף התכנית מחזורית, נשאל את השאלה - כמה זמן לוקח לתכנית לקחת איבר חדש מהוקטור ולהוסיף אותו לסכום?

בעזרת CPE נוכל לדעת כמה זמן ייקח, והאם אנו מנצלים את המעבד בצורה טובה

CPE

נשאף ל-CPE כמה שיותר נמוך

← בכל ריצה נשאף לפחות סיבובים (cycles)

← ככה התכנית תנצל את המשאבים של המעבד בצורה יותר טובה

Pentium III

ראינו בהרצאה דוגמה על חיבור איברי וקטור במעבד Pentium III

אילו פעולות אפשר לעשות במקביל במעבד הזה?

- load 1- הולכים לזיכרון וקוראים ערך ממנו
- store 1- הולכים לזיכרון ומאחסנים בו ערך
- integer 2- אפשר לעשות 2 פעולות שונות על מספרים שלמים
- FP addition 1- פעולה של חיבור וחיסור על מספרים לא שלמים
- FP multiplication and division 1- פעולה של כפל וחילוק על מספרים לא שלמים

Pentium III

מאיפה נובעת המגבלה? ישנן יחידות במעבד המוגדרות בדיוק לאותן פעולות (ואך ורק להן)

כלומר, ישנה יחידה אחת במעבד שמאפשרת לעשות load מהזברון, 2 יחידות במעבד שמאפשרות לעשות פעולות על מספרים שלמים, וכו'

נשים לב - חלק מהפעולות שציינו לוקחות יותר מ-cycle אחד

← אבל הן יכולות להתבצע כ-**pipeline**

pipeline

בפעולות כאלה- חלקים שונים של אותה פקודה מבוצעים ע"י רכיבים שונים במעבד

← לכן ניתן לחשב כמה רכיבים בו זמנית.

מה זה אומר? אפשר להתחיל פעולה אחת לפני שהשנייה הסתיימה באופן סופי

נראה כמה פעולות:

Latency - משך ביצוע הפעולה ביחידות של מחזורי שעון

Cycle - כל כמה מחזורי שעון אפשר להתחיל חישוב חדש

instruction	Latency	Cycles / Issue
Load / Store	3	1
Integer Add / Subtruct	1	1
Integer Multiply	4	1
Integer Division	36	36

נסתכל על איך תכניות שונות רצות בצורות שונות על המעבד, ועד כמה מהר התכנית יכולה לרוץ

התכנית הבאה כופלת איברים במערך
נראה את התכנית בשלב האחרון בהכנה לאסמבלי (מימין) ובאסמבלי (משמאל)

```
1  .L24:
2    imull (%eax,%edx,4),%ecx
3    incl %edx
4    cmpl %esi,%edx
5    jl .L24
```

```
loop:
    Multiply x by data[i]
    i++
    Compare i:length
    If <, goto loop
```

נסמן:
המערך data - %eax
המשתנה x - %ecx
המשתנה i - %edx
המשתנה length - %esi

```
1 .L24:
2   imull (%eax,%edx,4),%ecx
3   incl %edx
4   cmpl %esi,%edx
5   jl .L24
```

עכשיו נשאל- בהינתן פקודות באסמבלי- באילו execution units במעבד הן משתמשות?
כל פקודה באסמבלי יכולה להשתמש בכמה יחידות בתוך המעבד
למשל-

$$imull\ (\%eax, \%edx, 4), \%ecx$$

נשים לב שאנחנו רוצים קודם לגשת לזיכרון אז נשתמש בפעולת load,
ואח"כ תבוא יחידת הכפל imull

ככה נעשה עבור כל הפקודות באסמבלי ונקבל את התוצאה:

Assembly Instructions	Execution Unit Operations
.L24:	
imull (%eax,%edx,4),%ecx	load (%eax, %edx.0, 4) → t.1 imull t.1, %ecx.0 → %ecx.1
incl %edx	incl %edx.0 → %edx.1
cmpl %esi,%edx	cmpl %esi, %edx.1 → cc.1
jl .L24	jl-taken cc.1

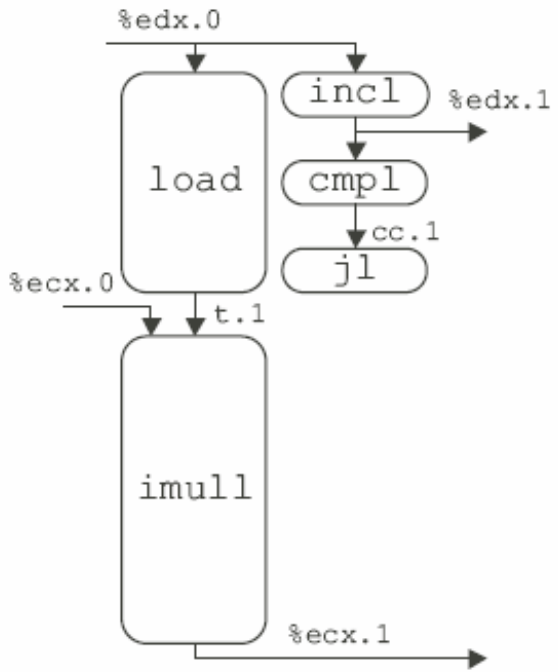
פיצלנו את המכפלה (שמחביאה גישה לזיכרון) ל Load ו mult

%eax - המערך data

%ecx - המשתנה x

%edx - המשתנה i

%esi - המשתנה length



אם נשים את זה בציור:

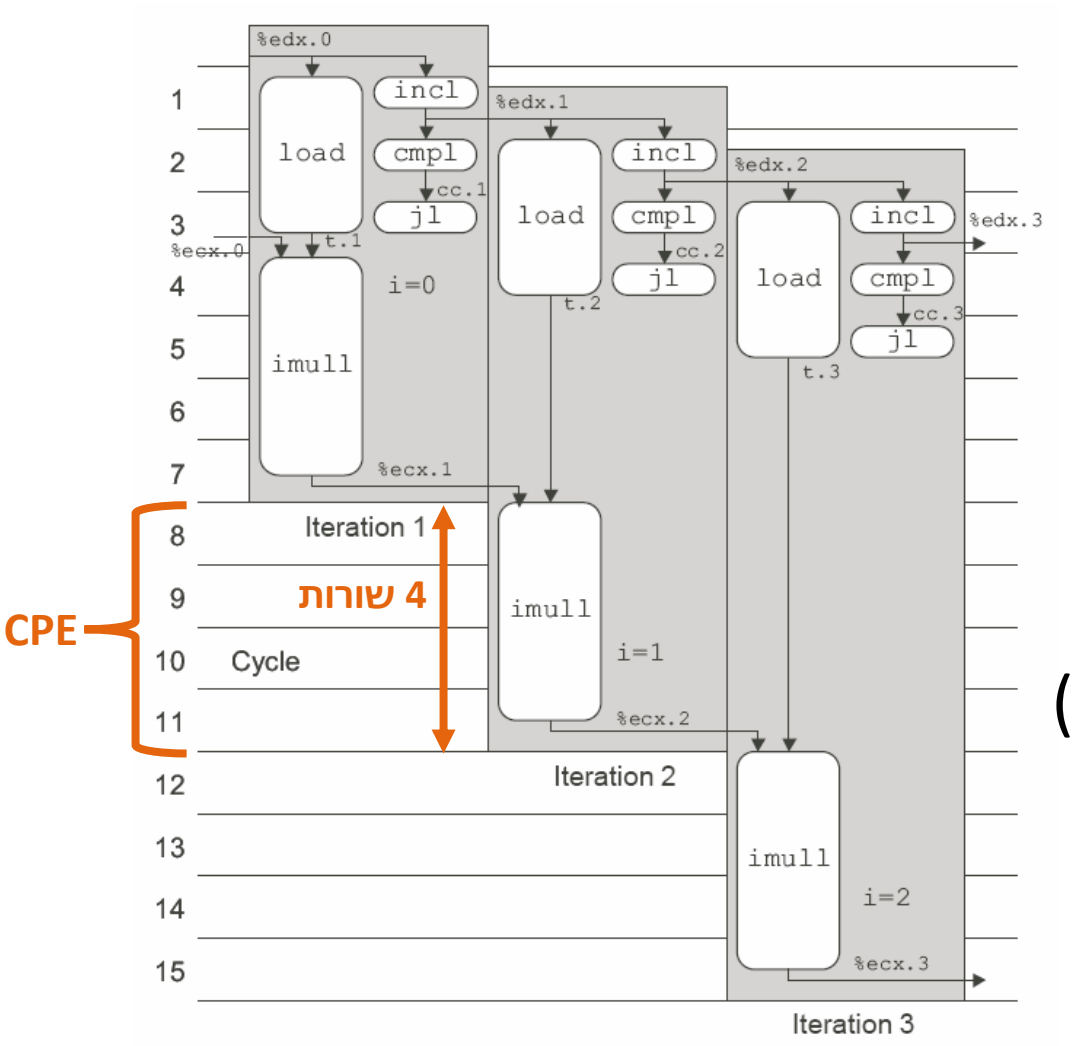
במה זמן לוקח לעבור על כל המערך?

בדוגמה הזו יש כמות בלתי מוגבלת של execution units

למרות זאת, נשים לב שה-cycle latency לא יכול לרדת מ-4

כדי לחשב את האיבר הבא במכפלה, צריך לחכות לאיבר הקודם (ונזכור שפעולת imull לוקחת 4 cycles)

לכן CPE=4



שיפור CPE ופריסת לולאות

אמרנו שאפשר לעשות 2 פעולות של integers במעבד

באיזו פונקציה ה- CPE עבור חישוב מכפלה יהיה קטן יותר?

```
int func_a(int* a, int size)
 {
        int result = 1;
        int i;
        for (i=0; i < size; i+=2)
         {
                result = result * a[i];
                result = result * a[i+1];
        }
        return result;
}
```

```
int func_b(int* a, int size)
 {
        int result1 = 1;
        int result2 = 1;
        int i;
        for (i=0; i < size; i+=2)
         {
                result1 = result1 * a[i];
                result2 = result2 * a[i+1];
        }
        return result1*result2;
}
```

שיפור CPE ופריסת לולאות

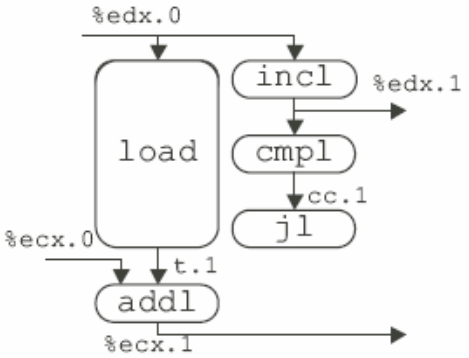
```
int func_b(int* a, int size)
 {
        int result1 = 1;
        int result2 = 1;
        int i;
        for (i=0; i < size; i+=2)
         {
                result1 = result1 * a[i];
                result2 = result2 * a[i+1];
        }
        return result1*result2;
}
```

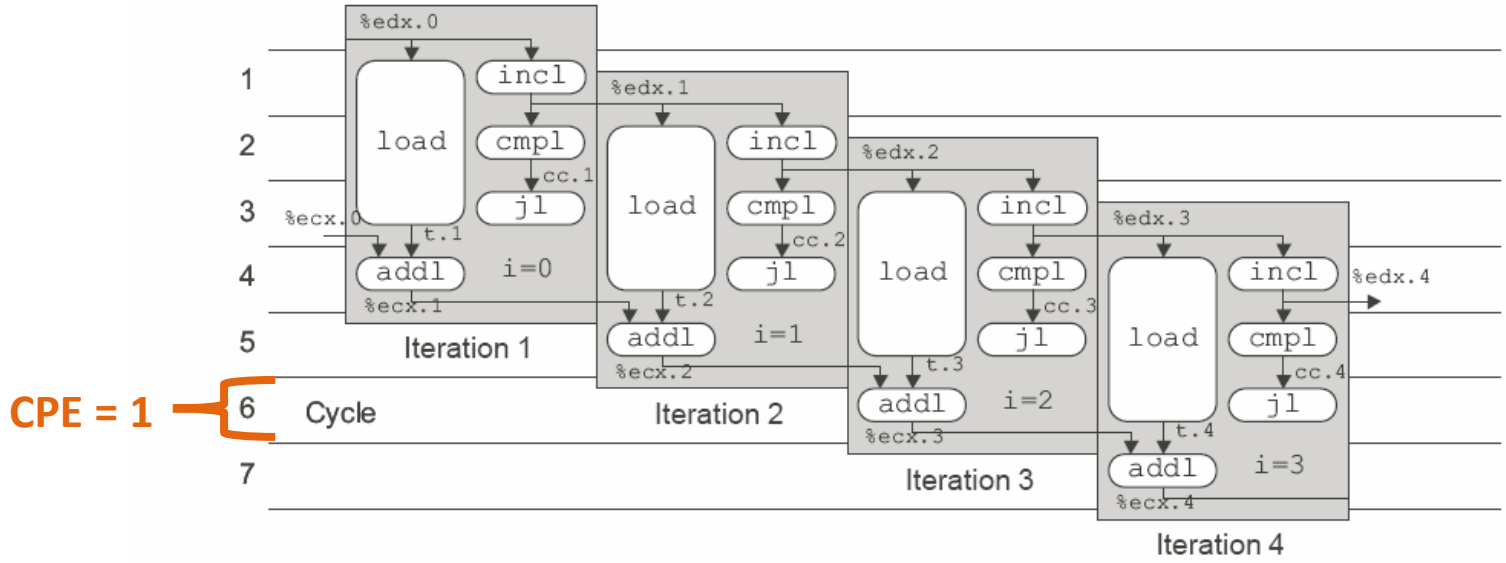
הפונקציה הזו (הימנית מהשקף הקודם) היא זו בעלת ה CPE הנמוך מבין השתיים. הסיבה לכך היא שה result לא תלויים אחד בשני וניתן לבצע את החישוב בצורה מקבילית.

באופן היפותטי, ניתן היה להמשיך לחלק לעוד ועוד מכפלות אבל בשלב מסוים זה כבר לא היה מוריד את ה CPE כי עדיין יש את המכפלה האחרונה.

פריסת לולאות-
סבימת איברי מערך ללא מגבלה על המשאבים

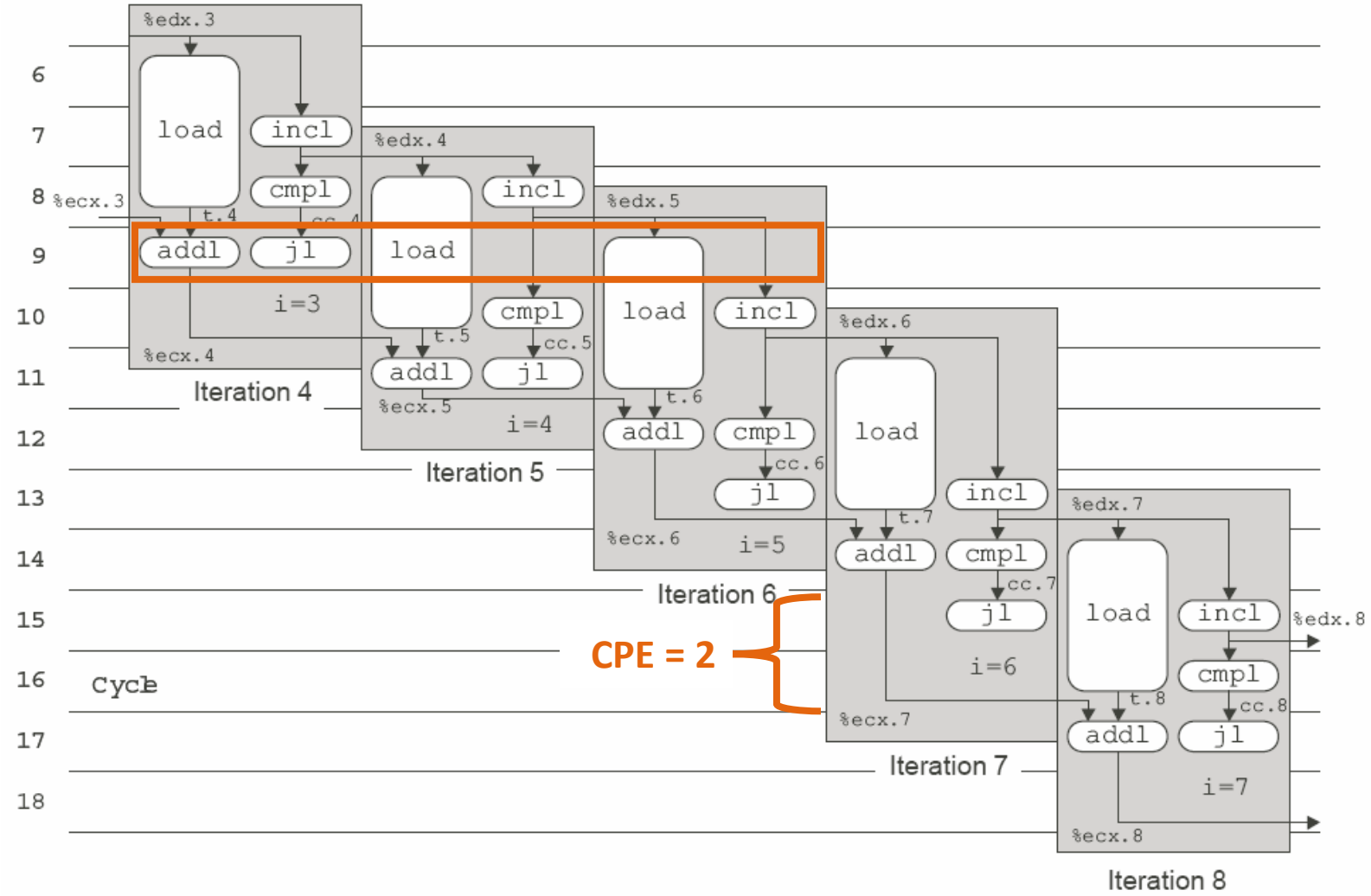
Execution Unit Operations		
load (%eax, %edx.0, 4)	→	t.1
addl t.1, %ecx.0	→	%ecx.1
incl %edx.0	→	%edx.1
cmpl %esi, %edx.1	→	cc.1
jl-taken cc.1		





מה ה CPE במקרה הזה?

פריסת לולאות-
סכימת איברי מערך עם מגבלה על המשאבים



איפה המגבלה באה לידי ביטוי?

שימו לב שבכל "שכבה" יש לכל היותר 2 פעולות על int.

CPE = 2

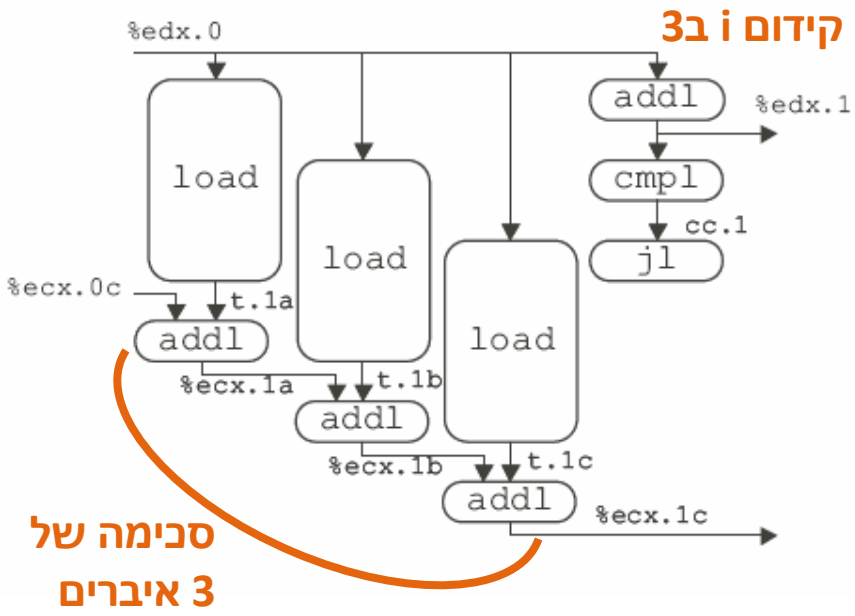
ביצד ניתן לשפר את ה-CPE כשיש מגבלות על המשאבים?

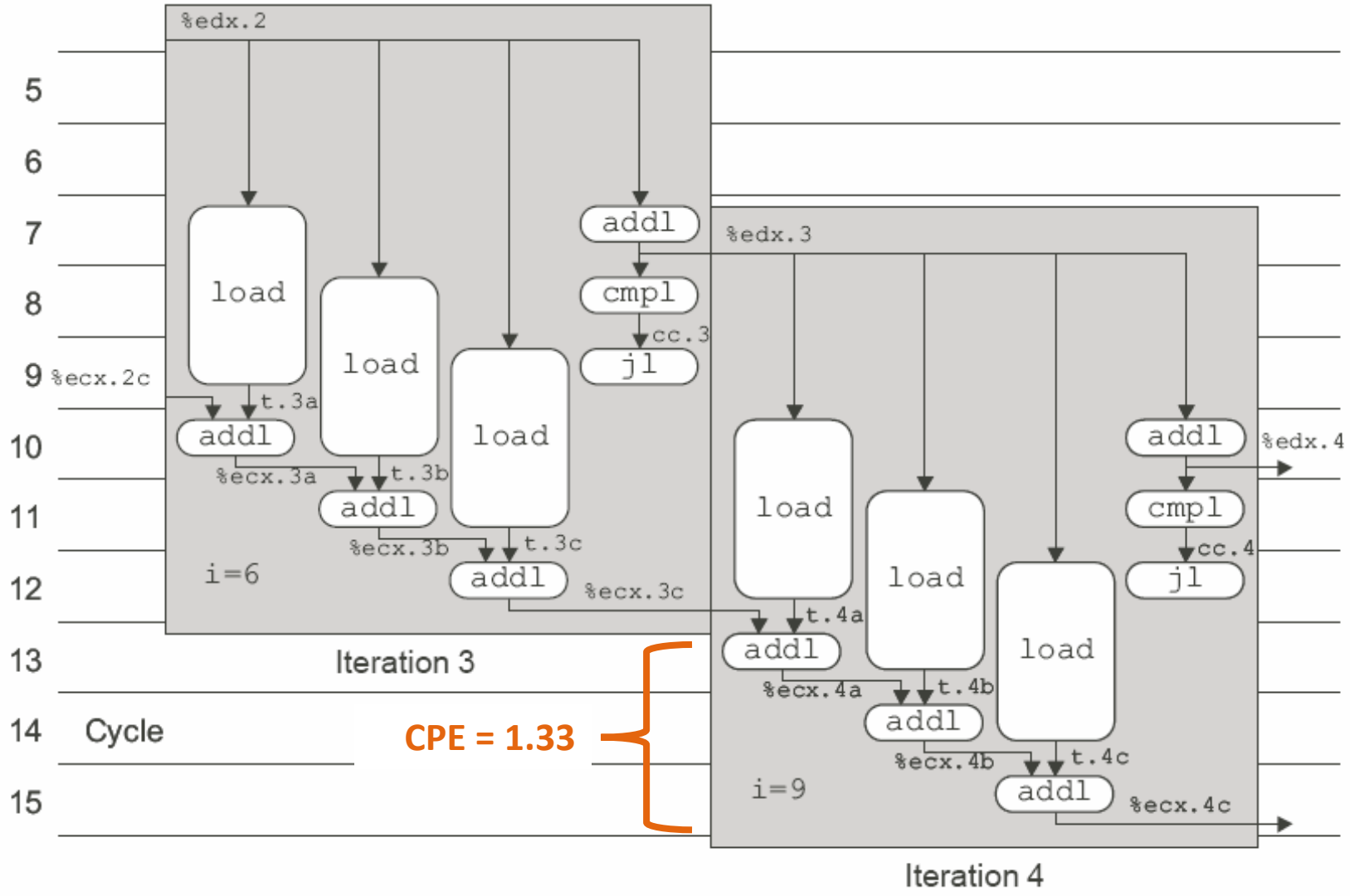
כמו בדוגמה עם הכפל- פורסים את הלולאה.

בדוגמה הזו- נפרוס ל-3

← מחשבים את החיבור ל-3 איברים ובסוף מקדמים את ה-i ב-3

Execution Unit Operations		
load (%eax, %edx.0, 4)	→	t.1a
addl t.1a, %ecx.0c	→	%ecx.1a
load 4(%eax, %edx.0, 4)	→	t.1b
addl t.1b, %ecx.1a	→	%ecx.1b
load 8(%eax, %edx.0, 4)	→	t.1c
addl t.1c, %ecx.1b	→	%ecx.1c
addl %edx.0, 3	→	%edx.1
cmpl %esi, %edx.1	→	cc.1
jl-taken cc.1		





לכאורה יכולנו להגיע

ל CPE = 1 אך לפי

החישובים שעשו

(ספר הקורס) הגיעו

ל CPE = 1.33

האם אפשר לעשות פריסה (unrolling) שתוציא תוצאה יותר טובה?

עשו מחקרים וגילו

Vector Length	Degree of Unrolling					
	1	2	3	4	8	16
CPE	2.00	1.50	1.33	1.50	1.25	1.06

בלומר, לפעמים ה-CPE עולה ולפעמים הוא יורד

תרגיל מסכם

הפונקציה הבאה מחשבת את המכפלה הפנימית של איברים ב-2 מערכים (בעלי אותו אורך)

```
int inner_product (int * u, int * v, int n)
{
	int i;
	int res=0;
	for ( i=0 ; i<n ; i++ )
	{
		res = res + u[i]*v[i];
	}
	return res;
}
```

המשימה: לחשב את ה-CPE עבור הפונקציה

שלב 1: מתרגמים לאסמבלי

```
inner_product:
move 8(R8), R3
move 12(R8), R2
move 16(R8), R6
xor R1,R1
xor R4,R4
.Loop:
        move (R2,R4,4),R5
        multiply (R3,R4,4),R5
        add R5,R1
        increment R4
        compare R6,R4
        jl .Loop
```

סימונים:

res -R1

R2- התחלת המערך u

R3- התחלת המערך v

i -R4

R5- רגיסטר ששומר את התוצאה ומעביר בסוף ל-R1

n -R6

שלב 2: חישוב הפקודות במעבד

inner_product:

.Loop:

move (R2,R4,4),R5

multiply (R3,R4,4),R5

add R5,R1

increment R4

compare R6,R4

jl .loop

load (R2,R4.0,4)	→	R5.1
load (R3,R4.0,4)	→	t.1
multiply t.1,R5.1	→	R5.1
add R5.1,R1.0	→	R1.1
increment R4.0	→	R4.1
compare R6,R4.1	→	cc.1
jl –taken cc.1		

שלב 3- מציירים איטרציה בודדת

load (R2,R4.0,4) → R5.1

load (R3,R4.0,4) → t.1

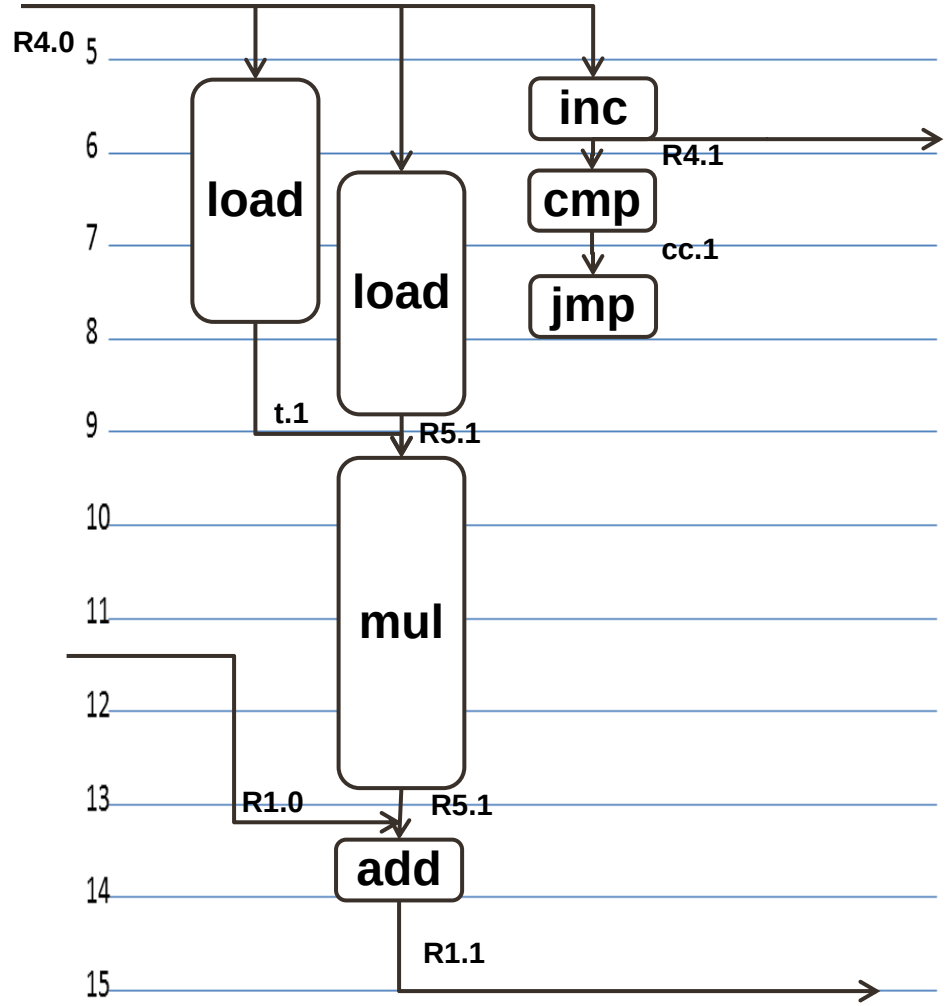
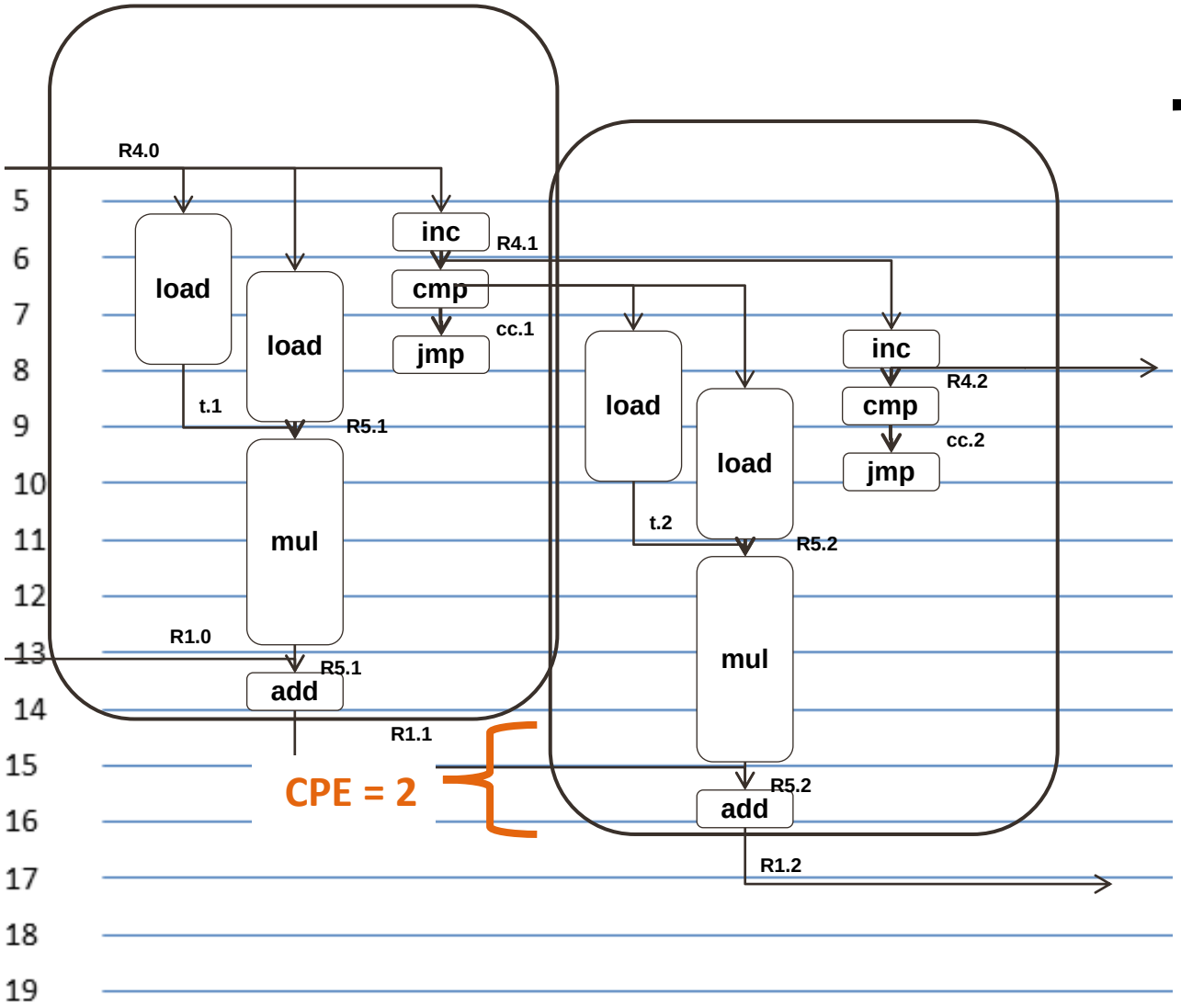
multiply t.1,R5.1 → R5.1

add R5.1,R1.0 → R1.1

increment R4.0 → R4.1

compare R6,R4.1 → cc.1

jl –taken cc.1



שלב 4: חישוב ה-CPE