

Out Of Order Execution

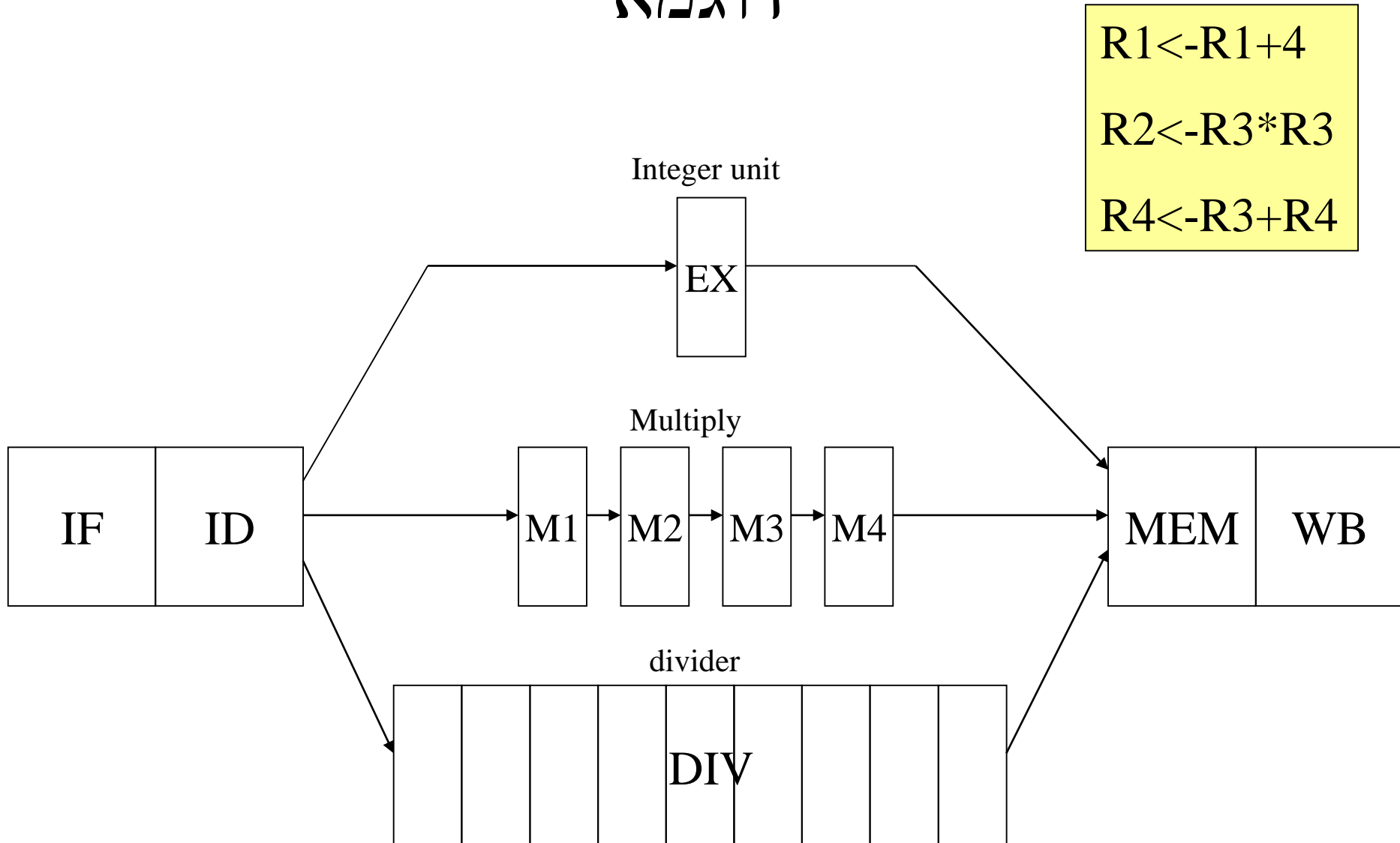
(Part 1)

Updated by Franck Sala

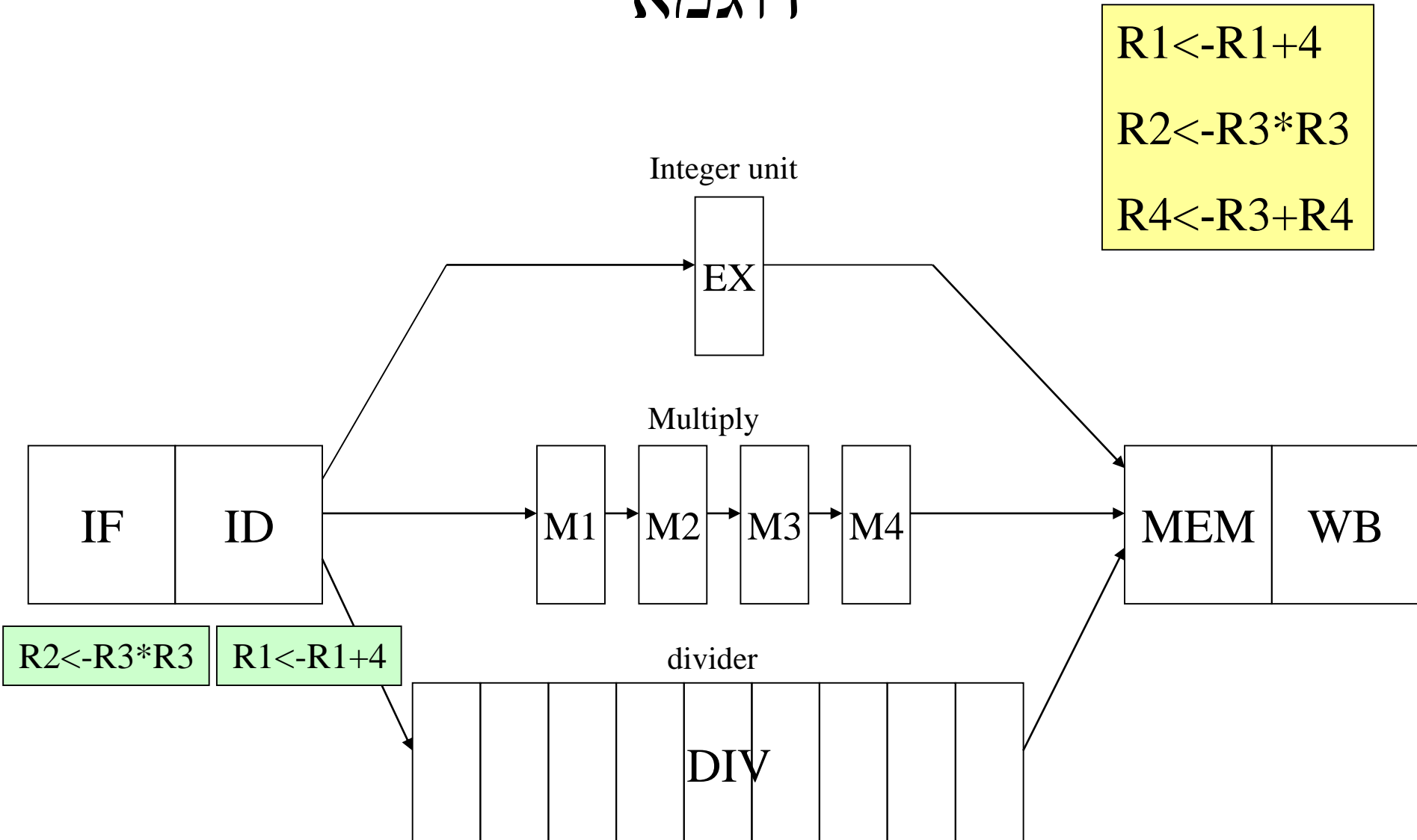
Execution of instructions with variable execution time

- The longest operation in the machine fixes the frequency
- Break long instruction in many short operations
 - Implement a pipeline in EXE
 - Not possible for all the instructions
 - Execution time of certain instruction is variable (load with cache miss...)
 - More pipe stages = bigger penalty on misprediction and more data hazards (CPI increase)

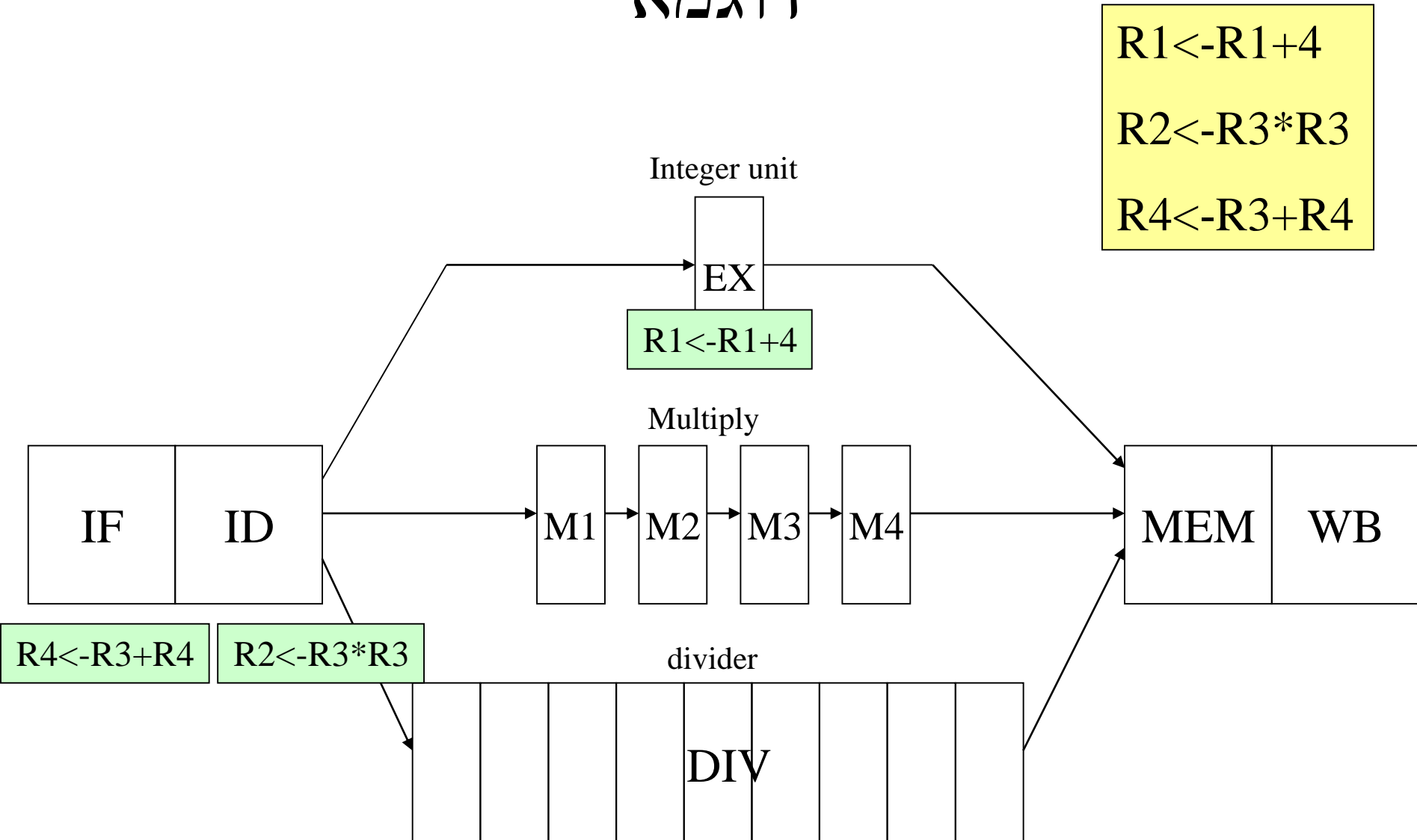
דוגמא



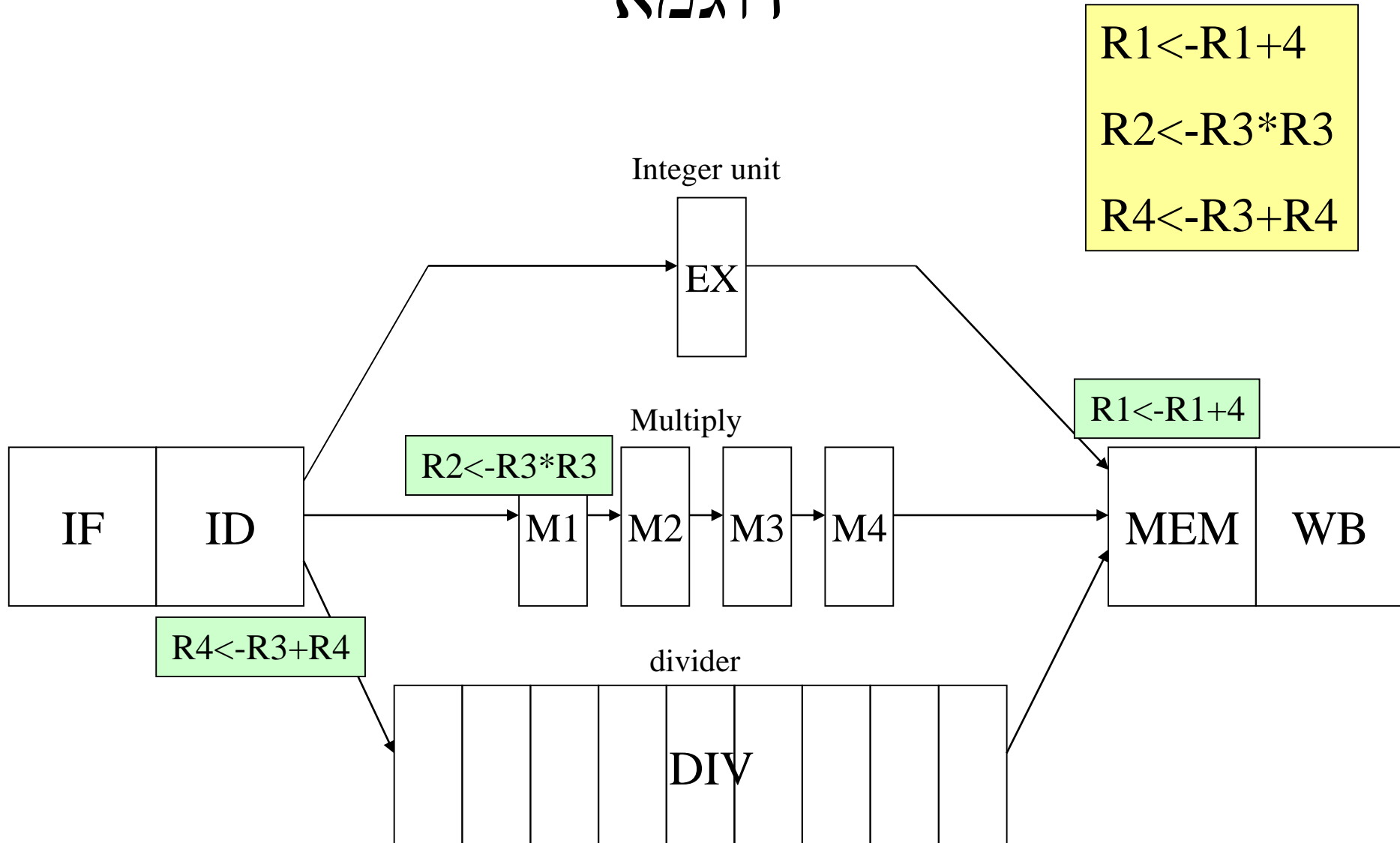
דוגמא



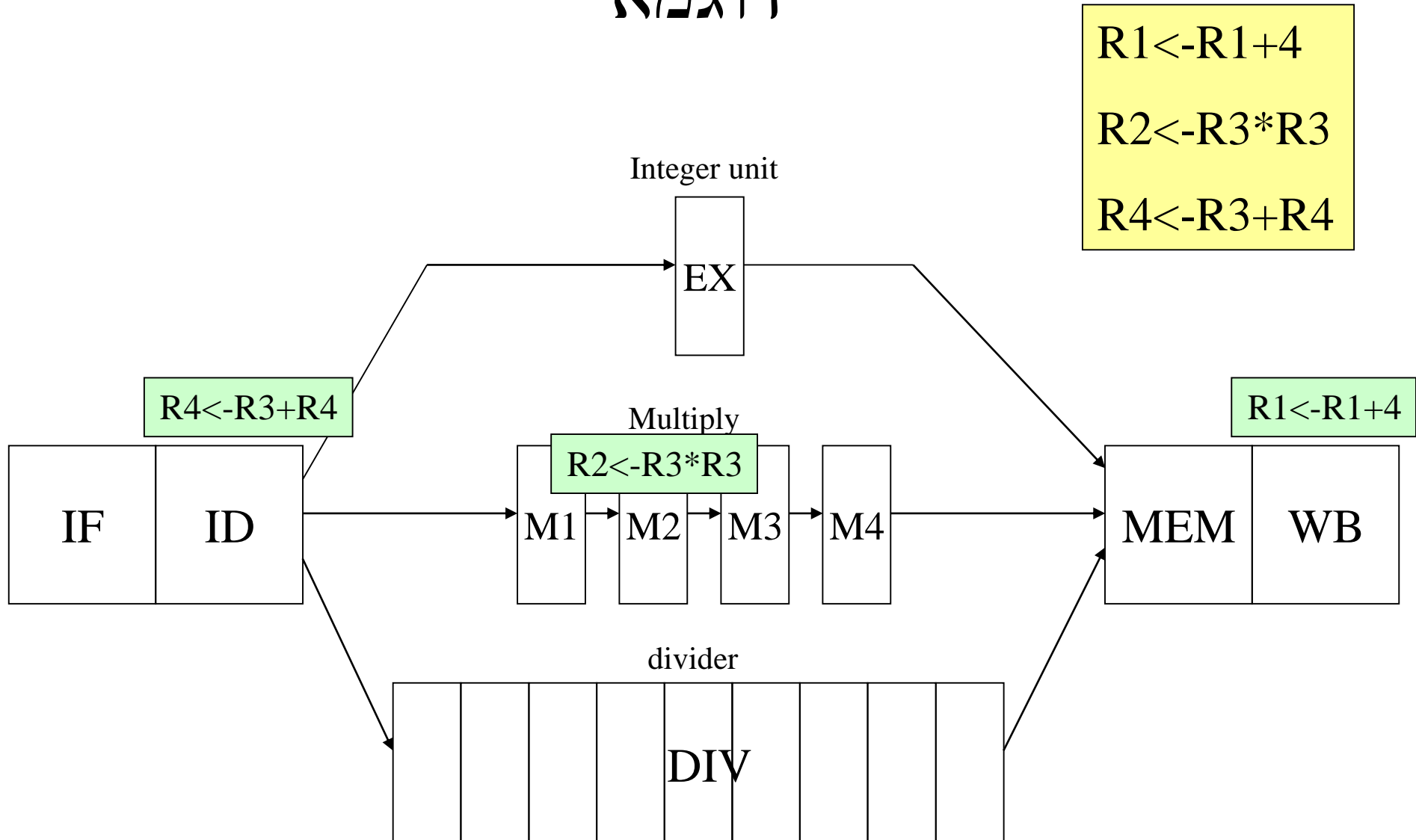
דוגמא



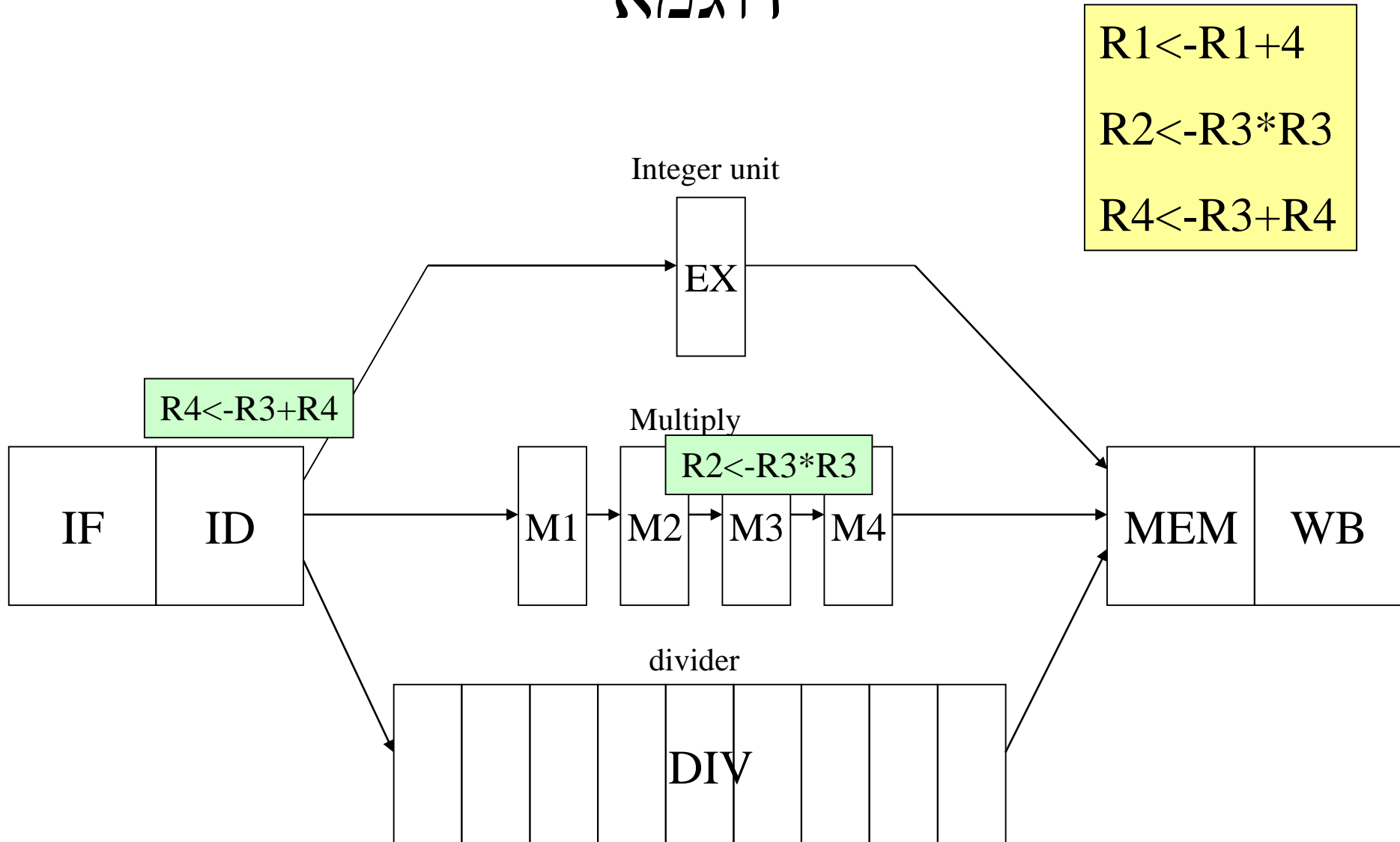
דוגמא



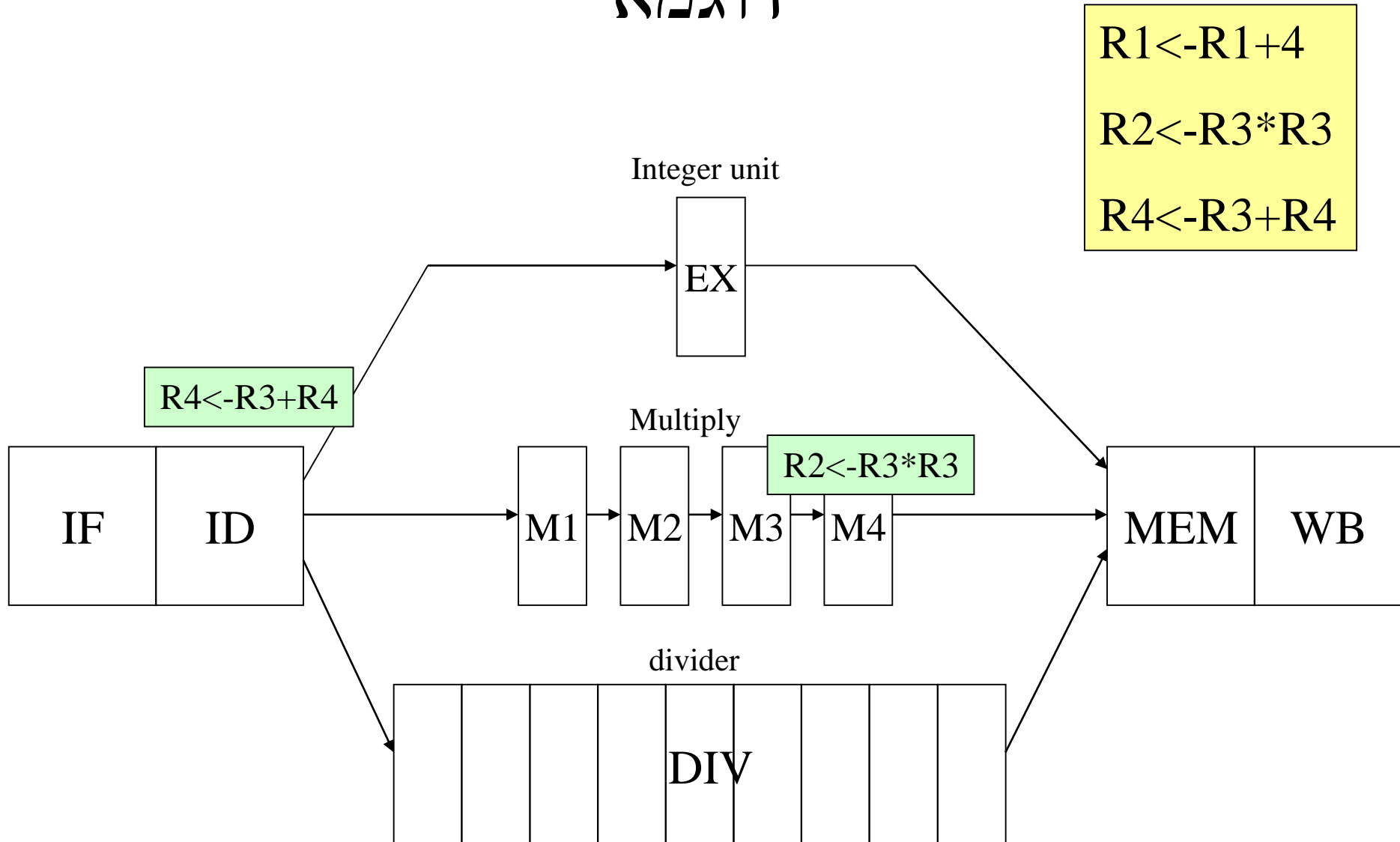
דוגמא



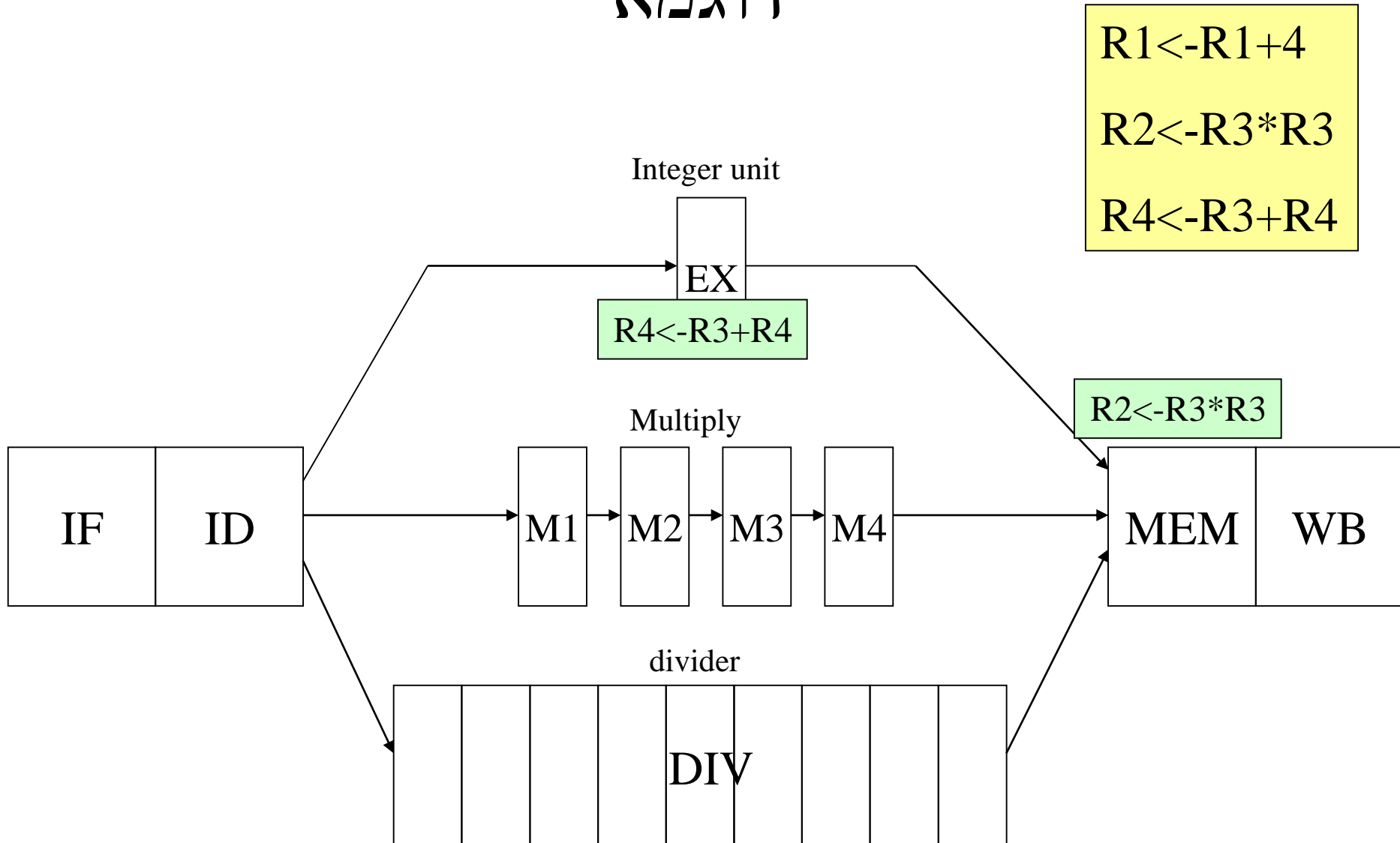
דוגמא



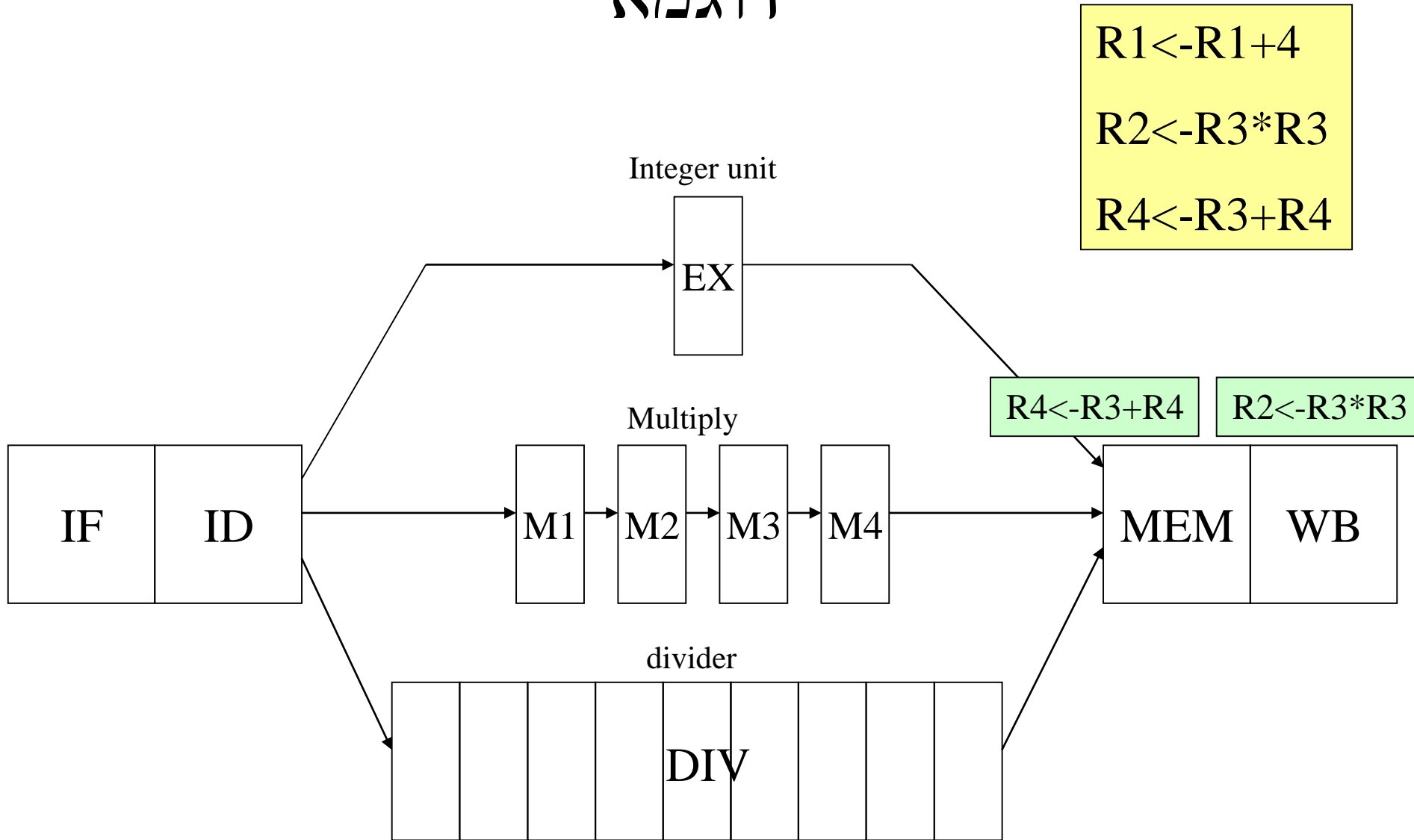
דוגמא



דוגמא



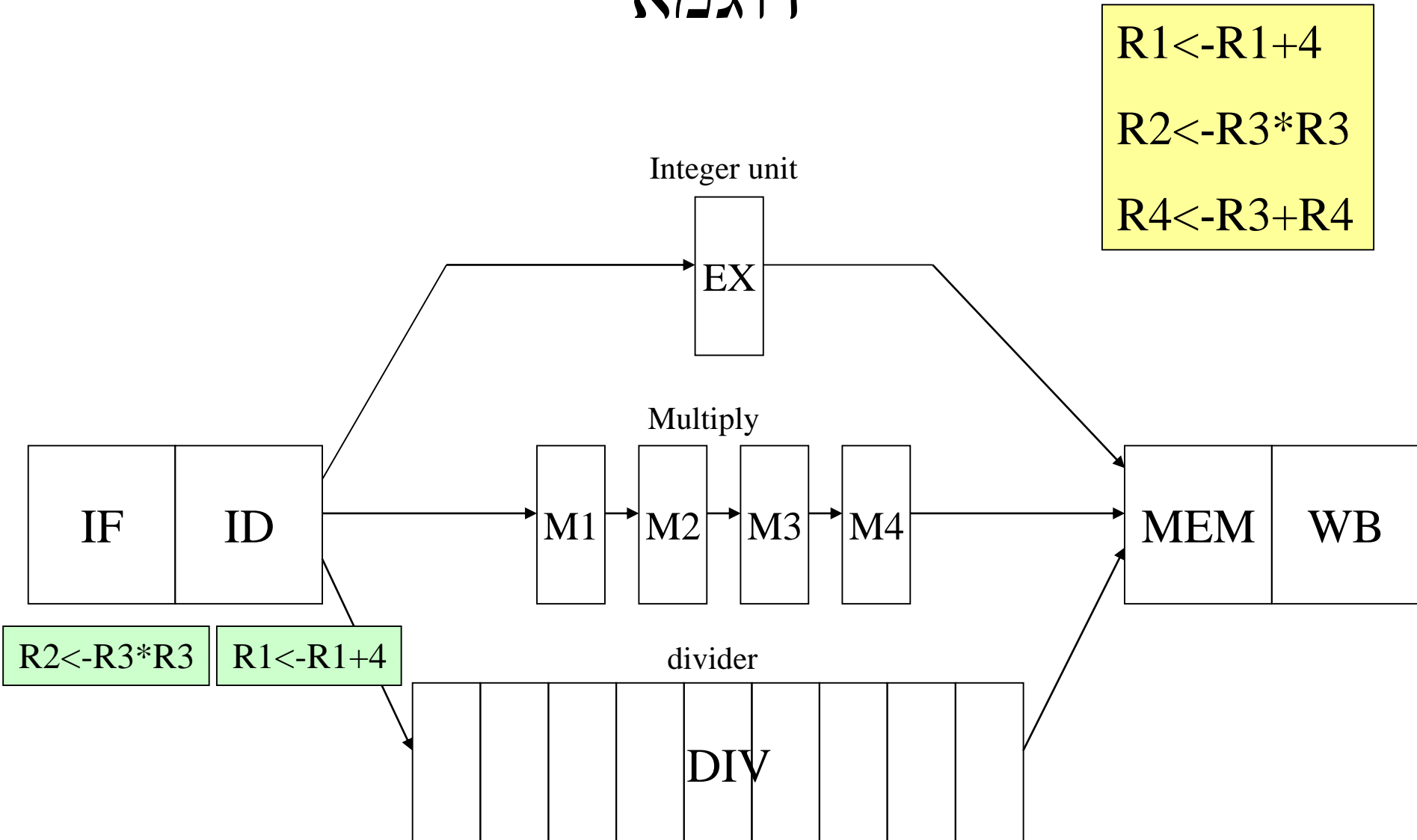
דוגמא



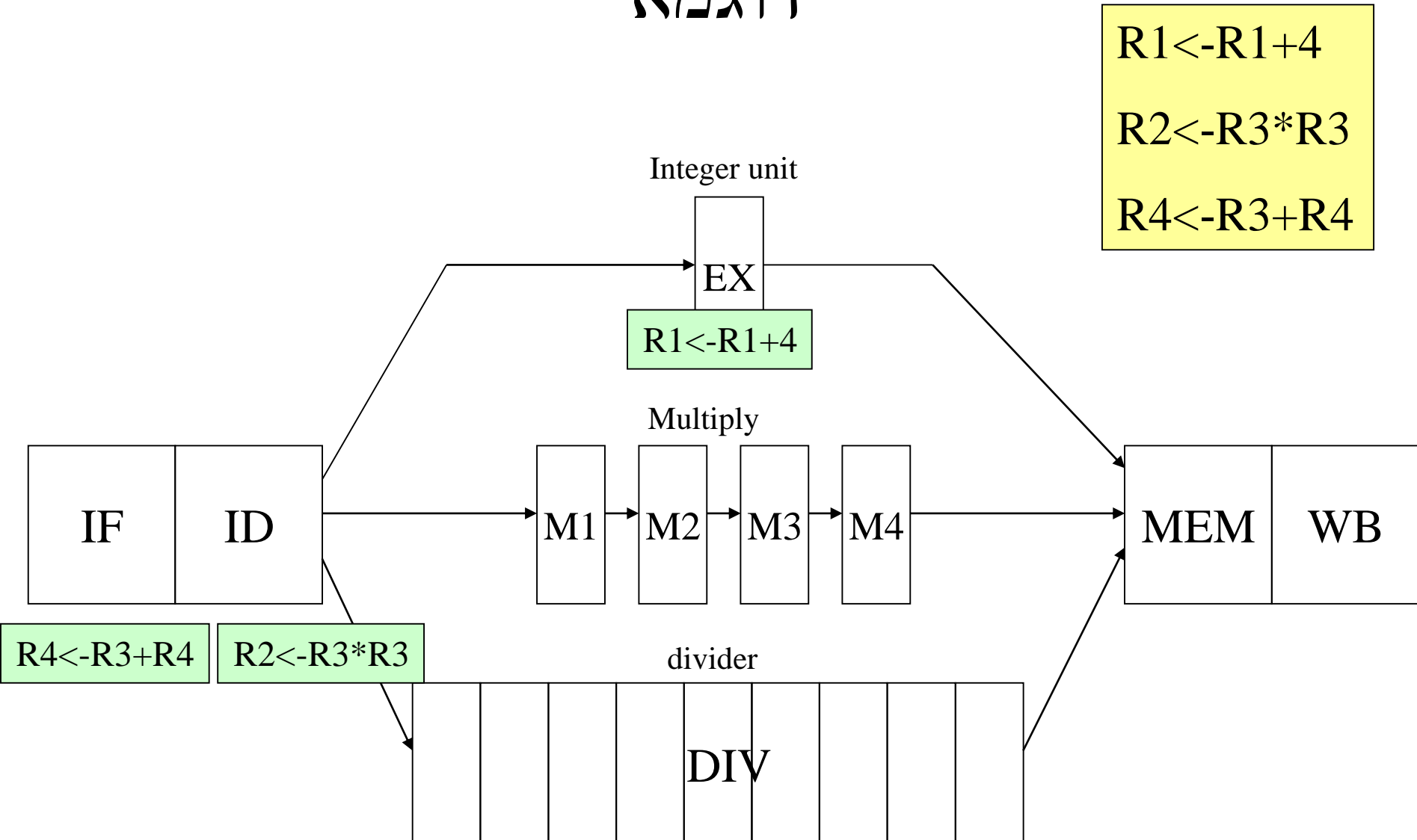
Execution of instructions with variable execution time

- Allow instructions of different pipelines to be executed Out of Order
 - Execute many **independent** instructions in parallel in different pipelines
 - Execution must keep correctness of the code
 - Improve CPI

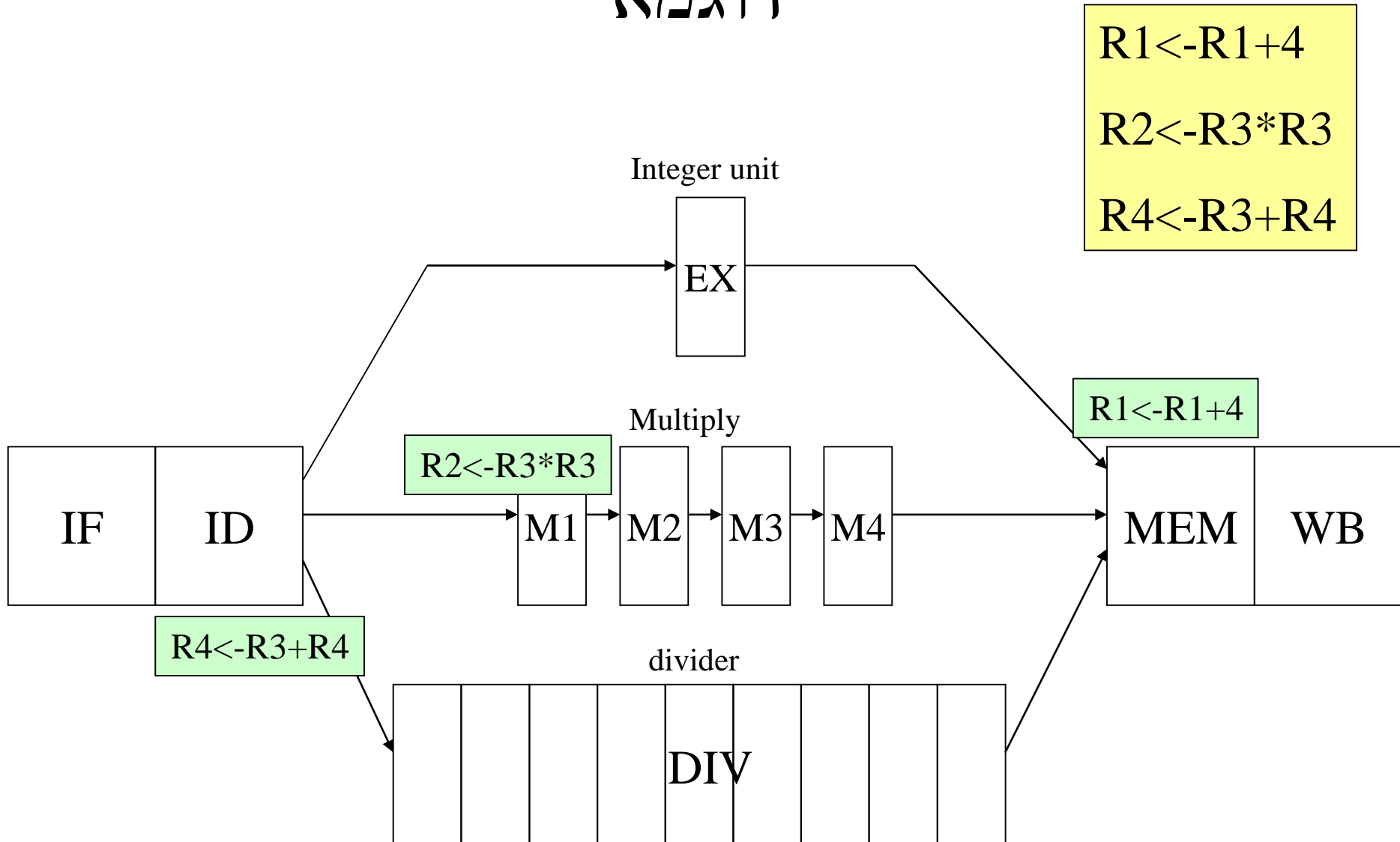
דוגמא



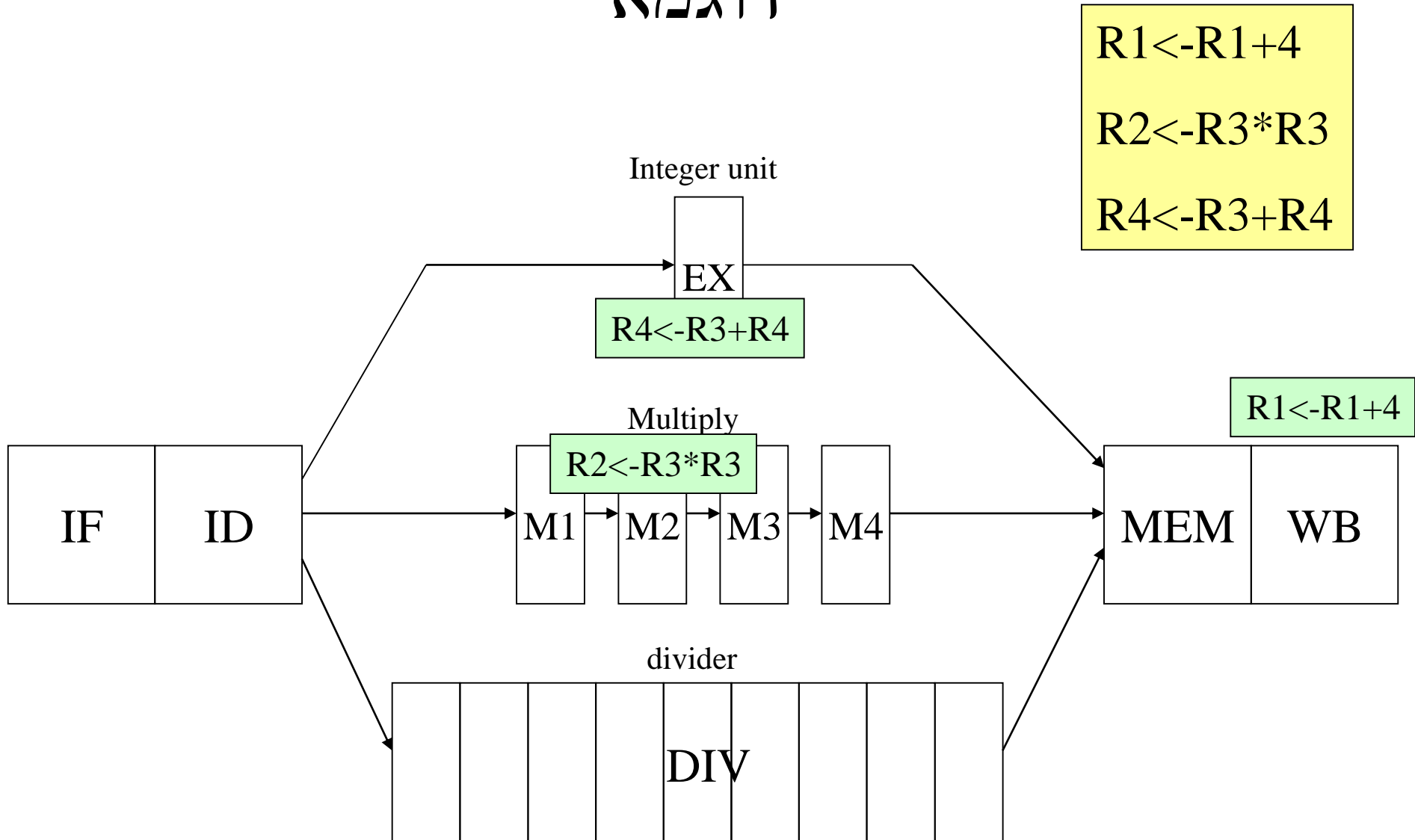
דוגמא



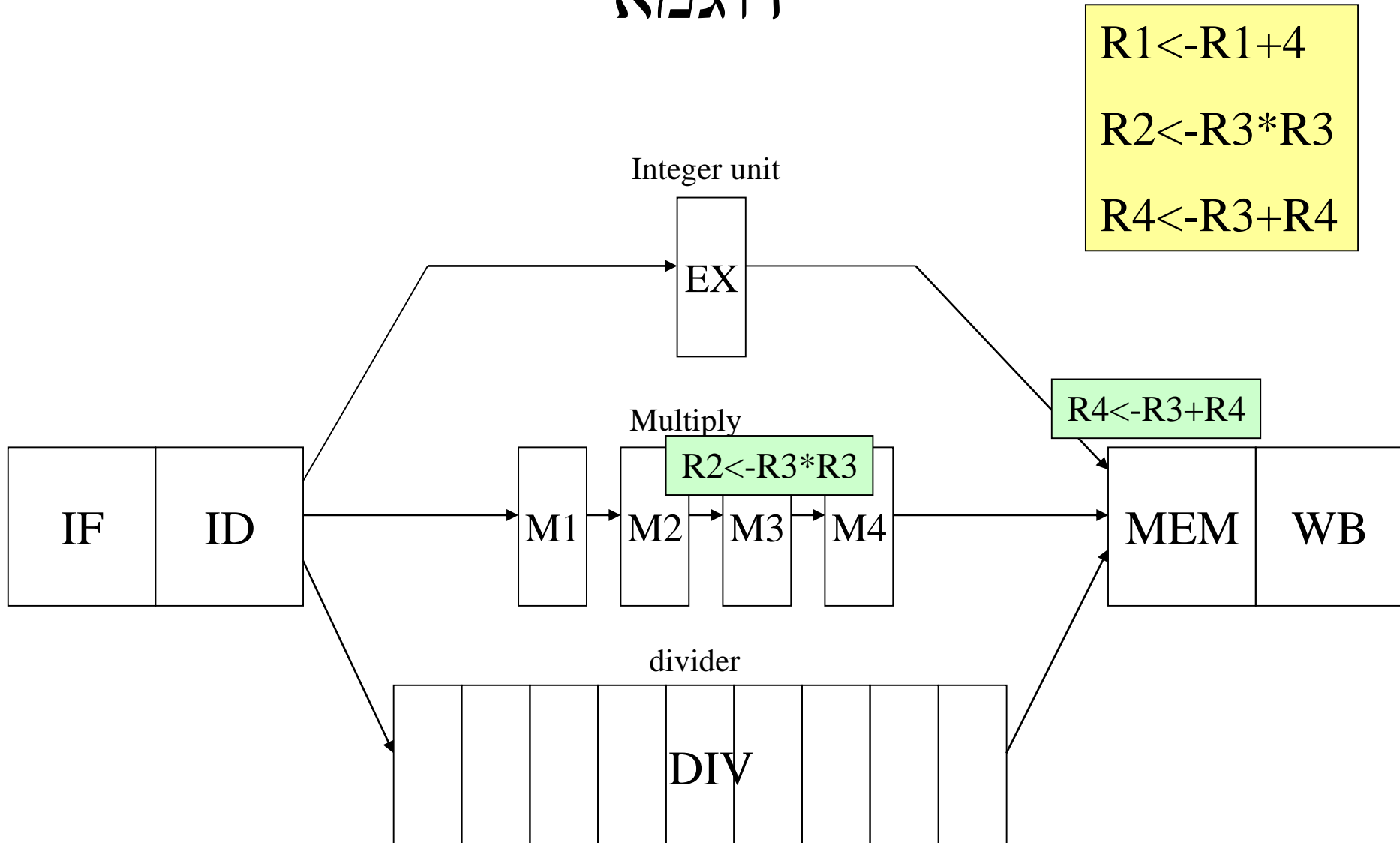
דוגמא



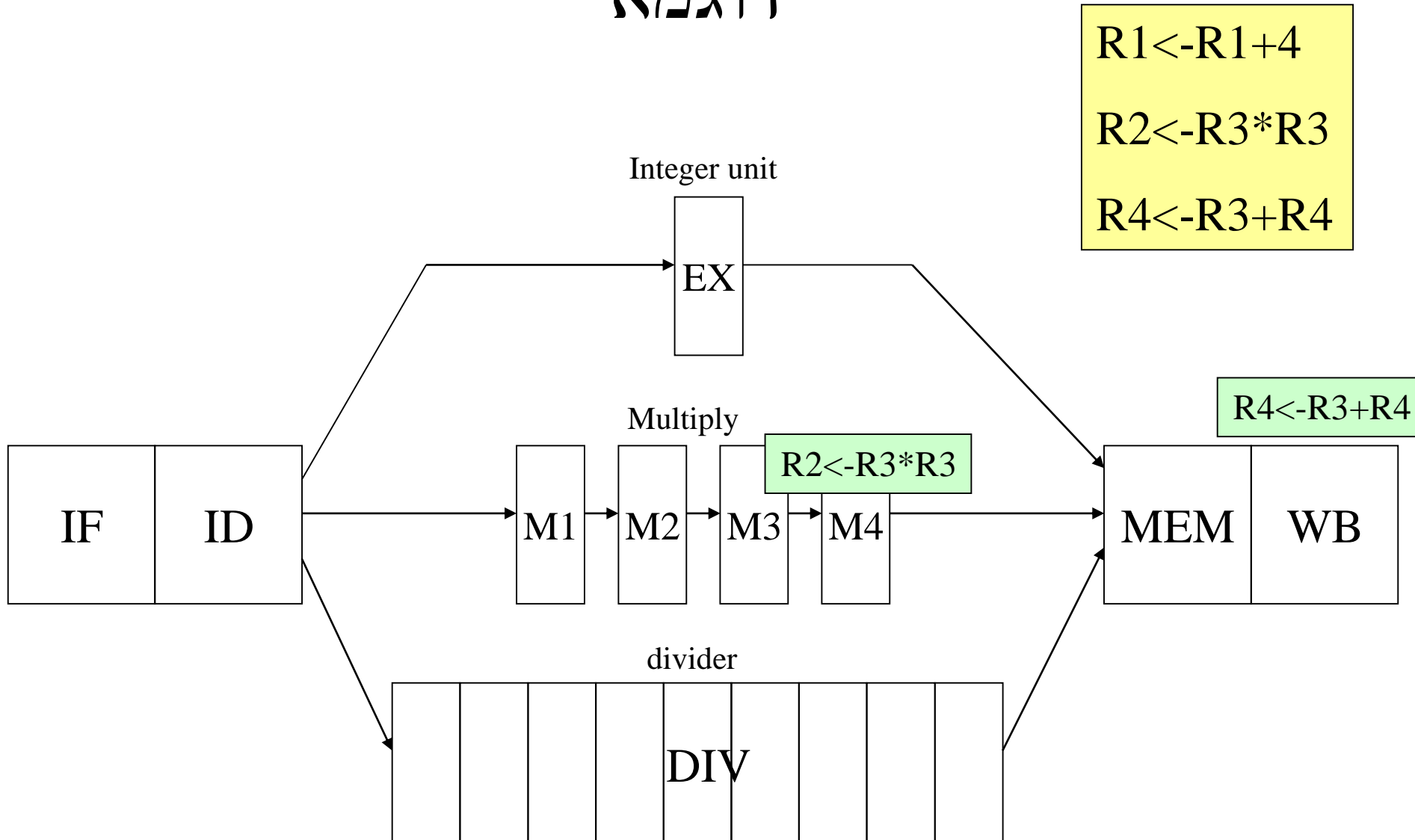
דוגמא



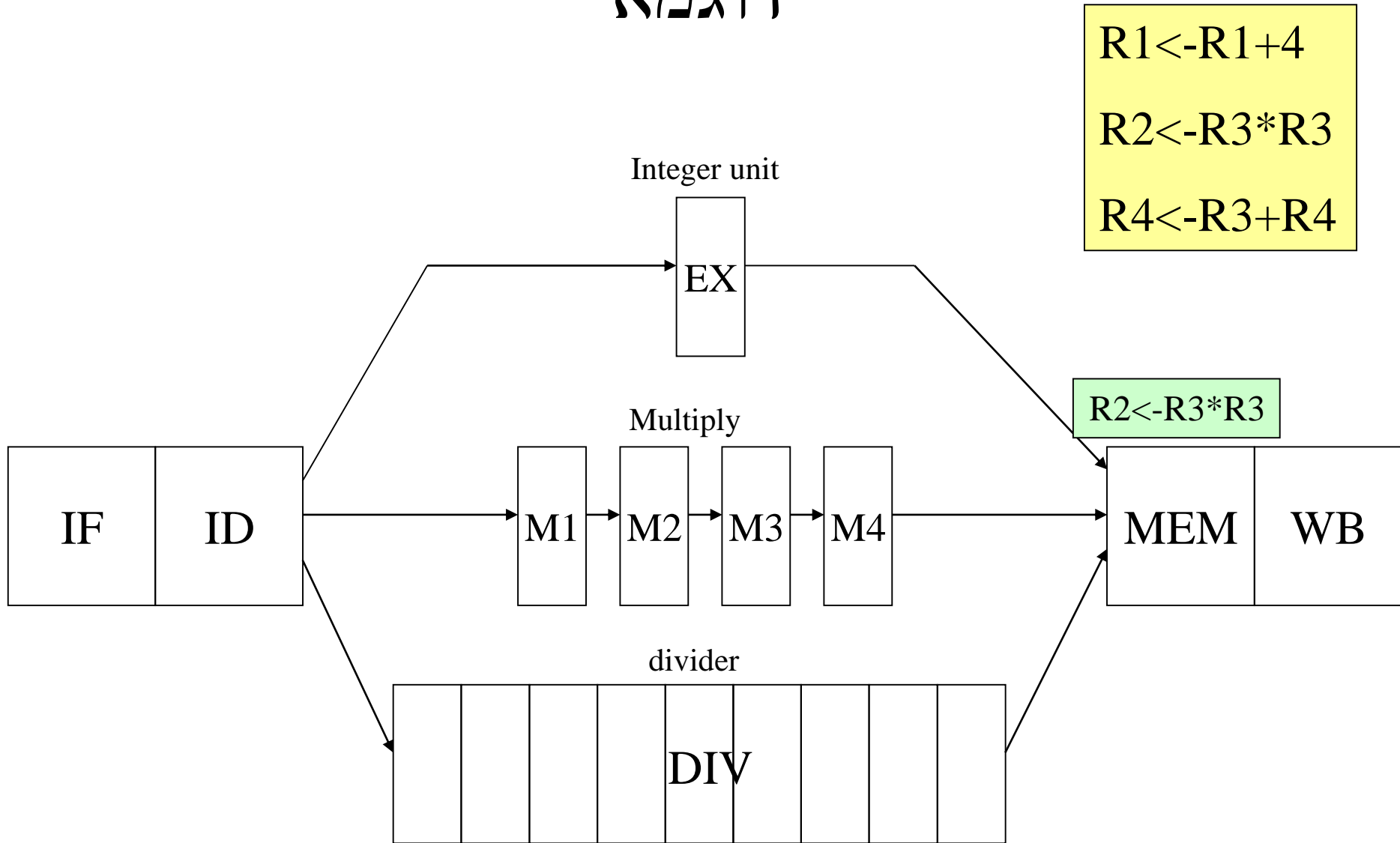
דוגמא



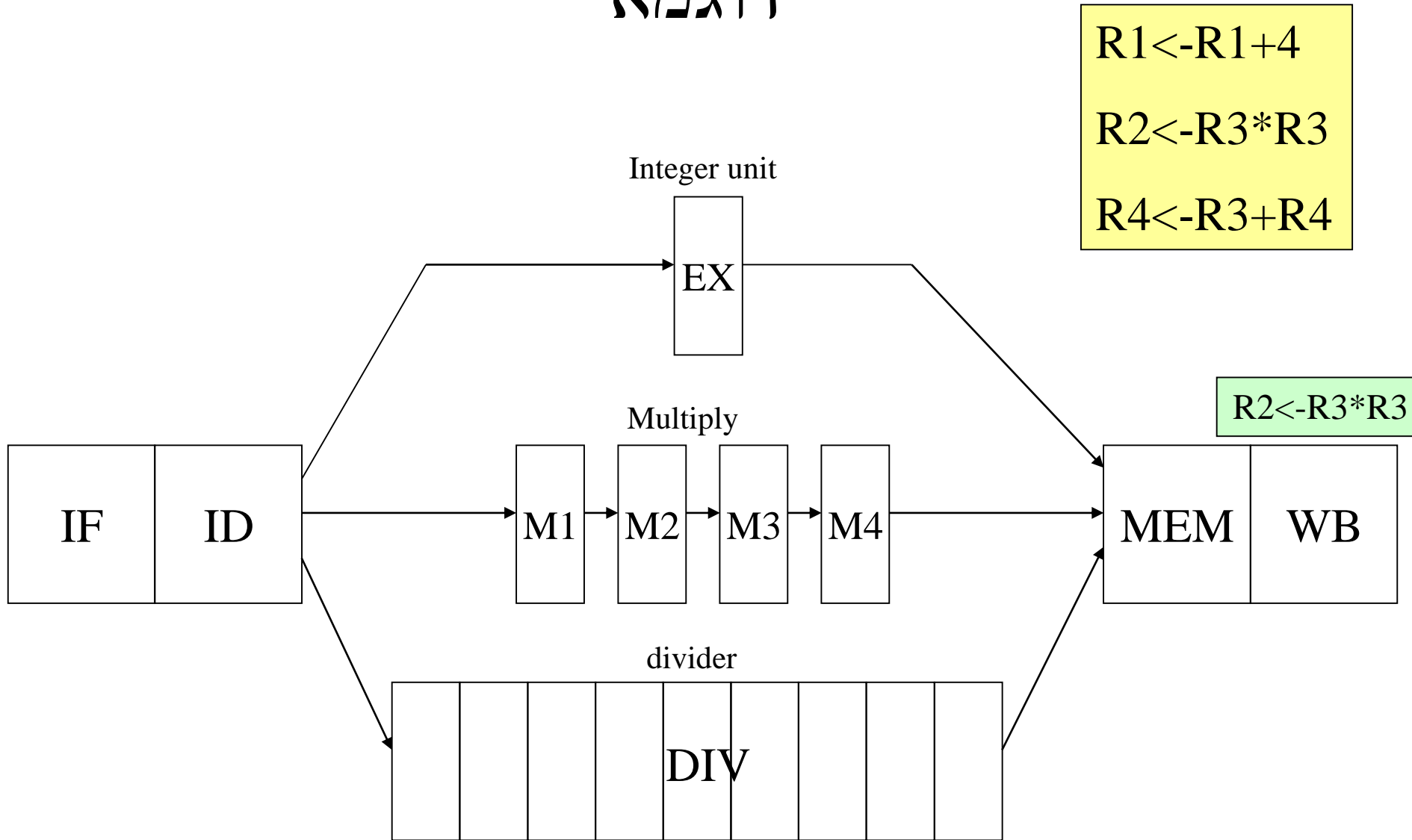
דוגמא



דוגמא

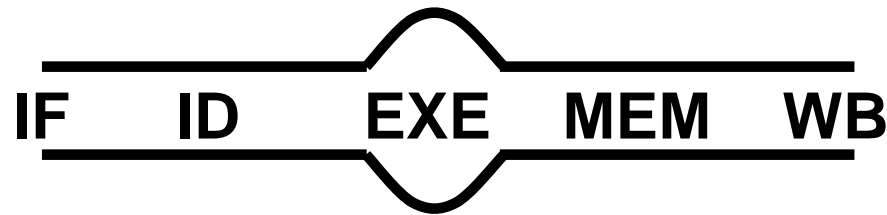


דוגמא

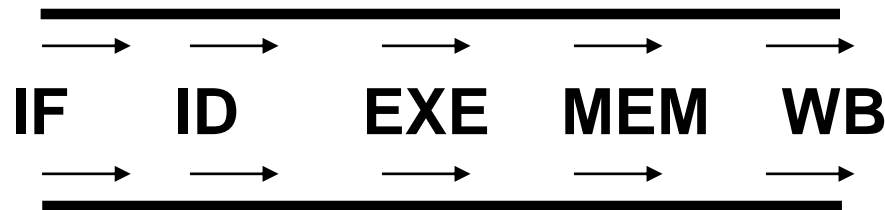


CPI

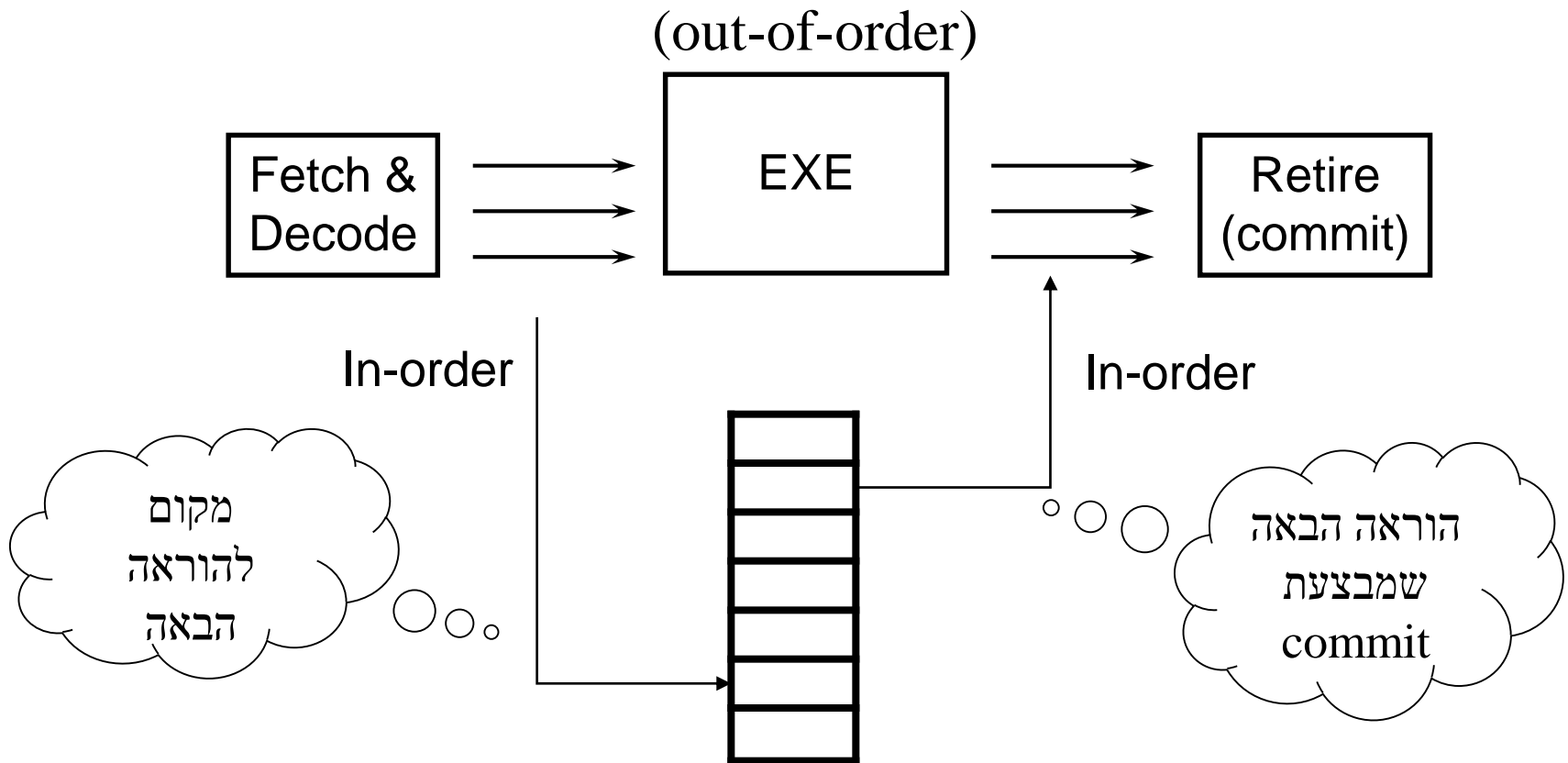
- עניין נוסף: במכונות **in order** כמו שהכרנו עד היום ניתן (אם בכלל) להגיע ל-CPI מינימאלי של 1
- גם בביצוע של OOOE יש את חסם זה אם נאפשר ביצוע מקביל רק של שלב ה-EXE.



- אבל נאפשר גם בשאר השלבים ביצוע מספר הוראות במקביל אזי שנוכל לשבור את חסם זה.



OOO-Execution Processor



תלויות מידע (data hazard)

- להזכירכם אנו מכירים כבר תלות מידע מסוג **RAW**:

ADD **R1**, R2, R3
ADD R5, R6, **R1**



- **OOOE** יוצרת תלויות מידע חדשות:

WAR – Write After Read ❖

DIV R1, R2, R3 // *many cycles*

ADD R5, **R6**, R1 // *depends on previous instruction*

ADD **R6**, R7, R8 // *WAR issue*

- הוראת החילוק הארוכה מעכבת את הוראת החיבור העוקבת שממתינה לערך של R1, מכיוון שאנו מאפשרים ביצוע out of order ההוראה השלישית שלא מחכה לכולם תתבצע, ואז ההוראה השנייה לכשתתבצע עלולה לקרוא את הערך של R6 כפי שנכתב ע"י ההוראה השלישית ולא כפי שהתכוון המשורר (או המתכנת).

❖ **WAW** – Write After Write

DIV R1,R2,R3
ADD R5,R6,R1
ADD R5,R7,R8

// many cycles

// depends on previous instruction

// WAW issue

- גם כאן ההוראה השנייה מעוכבת ואילו השלישית לא, מה שעלול להתבצע הוא שההוראה השלישית תכתוב ל-R5 לפני ההוראה השנייה, ולבסוף כשהשנייה תכתוב גם כן, היא תשנה את ערך זה. בסופו של דבר, יחזיק R5 בערך לא עדכני.

❑ תלויות אלו נקראות **False Dependencies** (כיוון שאם היה לרשות המהדר (קומפילר) מספר בלתי מוגבל של רגיסטרים היה יכול לבחור רגיסטרים שונים לכל פעולה ואז לא היו הפקודות תלויות אחת בשנייה (יותר)

❑ בעיה נוספת שעוד לא הכרנו היא **Structural Hazard**, זהו מצב בו הוראה כלשהי נאלצת להמתין כי אין יחידת חישוב פנויה עבורה. למשל רצף פעולות חילוק עלול לגרום לכך.

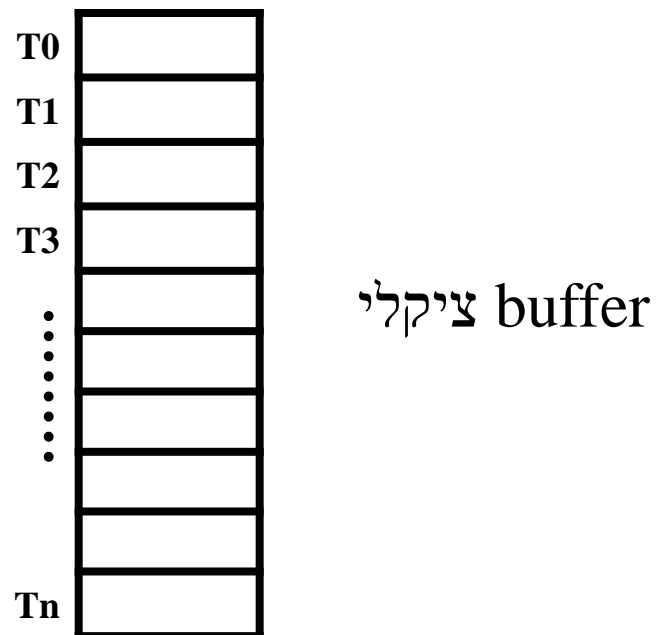
Register Renaming

- דרך לפתור את בעיית התלויות החדשות היא ע"י **Register Renaming**.
- הרעיון: שמירת שתי מערכות רגיסטרים זאת אשר גלויה למשתמש (**רגיסטרים ארכיטקטוניים**) ואשר בשימוש המהדר (שפת הסף), ומספר גדול של **רגיסטרים פיזיים** מתוך מאגר (pool)
- **אנו מבצעים שני מיפויים**:
 - האחד מהרגיסטרים הארכיטקטוניים לרגיסטרים הפיזיים לצורך פתירת בעיות תלויות מדומות. המיפוי הזה מבוצע במעבר בין שלב ה- **decode** לשלב הביצוע
 - והשני מהרגיסטרים הפיזיים לארכיטקטוניים לצורך שמירת "תמונת המכונה" כפי שהקומפילר בנה אותה. המיפוי הזה מבוצע כחלק מפעולת **commit** ומעבירים את הערך שלהם לרגיסטרים המקוריים
- ע"פ רוב, מעבדים מודרניים מבצעים את שלבי ה- **commit** וה- **fetch** **decode** בסדר התוכנית (in order) ואילו את שלב הביצוע מבצעים out of order.

Arch reg.	Temp reg.
R1	
R2	
R3	
R4	

Renamer

דוגמא:

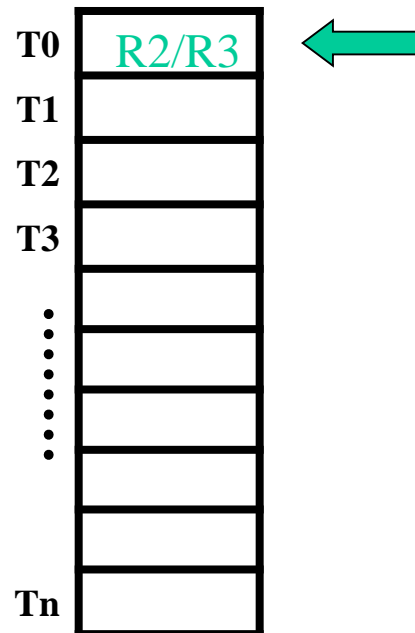


Arch reg.	Temp reg.
R1	T0
R2	
R3	
R4	

Renamer

:וגמא

DIV R1,R2,R3
 ADD R2,R4,R1
 ADD R2,R3,R3
 ADD R4,R3,R2

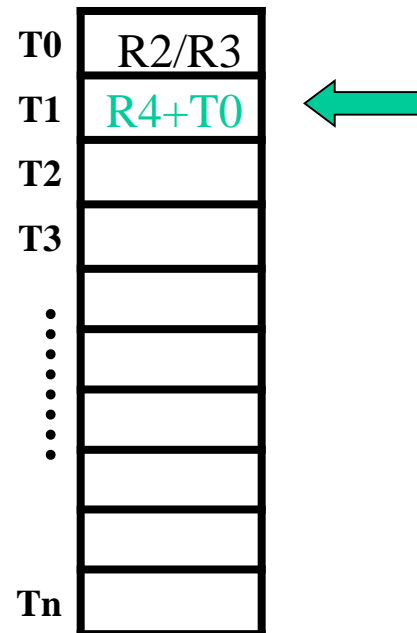


Arch reg.	Temp reg.
R1	T0
R2	T1
R3	
R4	

Renamer

:וגמא

DIV R1,R2,R3
 ADD R2,R4,R1
 ADD R2,R3,R3
 ADD R4,R3,R2

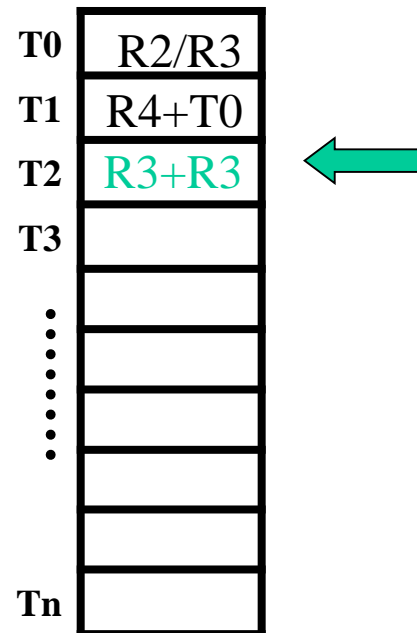


Arch reg.	Temp reg.
R1	T0
R2	T2
R3	
R4	

Renamer

:וגמא

DIV R1,R2,R3
 ADD R2,R4,R1
 ADD R2,R3,R3
 ADD R4,R3,R2



Arch reg.	Temp reg.
R1	T0
R2	T2
R3	
R4	T3

Renamer

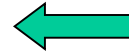
דוגמא:

```

DIV  R1,R2,R3
ADD  R2,R4,R1
ADD  R2,R3,R3
ADD  R4,R3,R2

```

T0	R2/R3
T1	R4+T0
T2	R3+R3
T3	R3+T2
⋮	
⋮	
⋮	
⋮	
Tn	



שים לב:

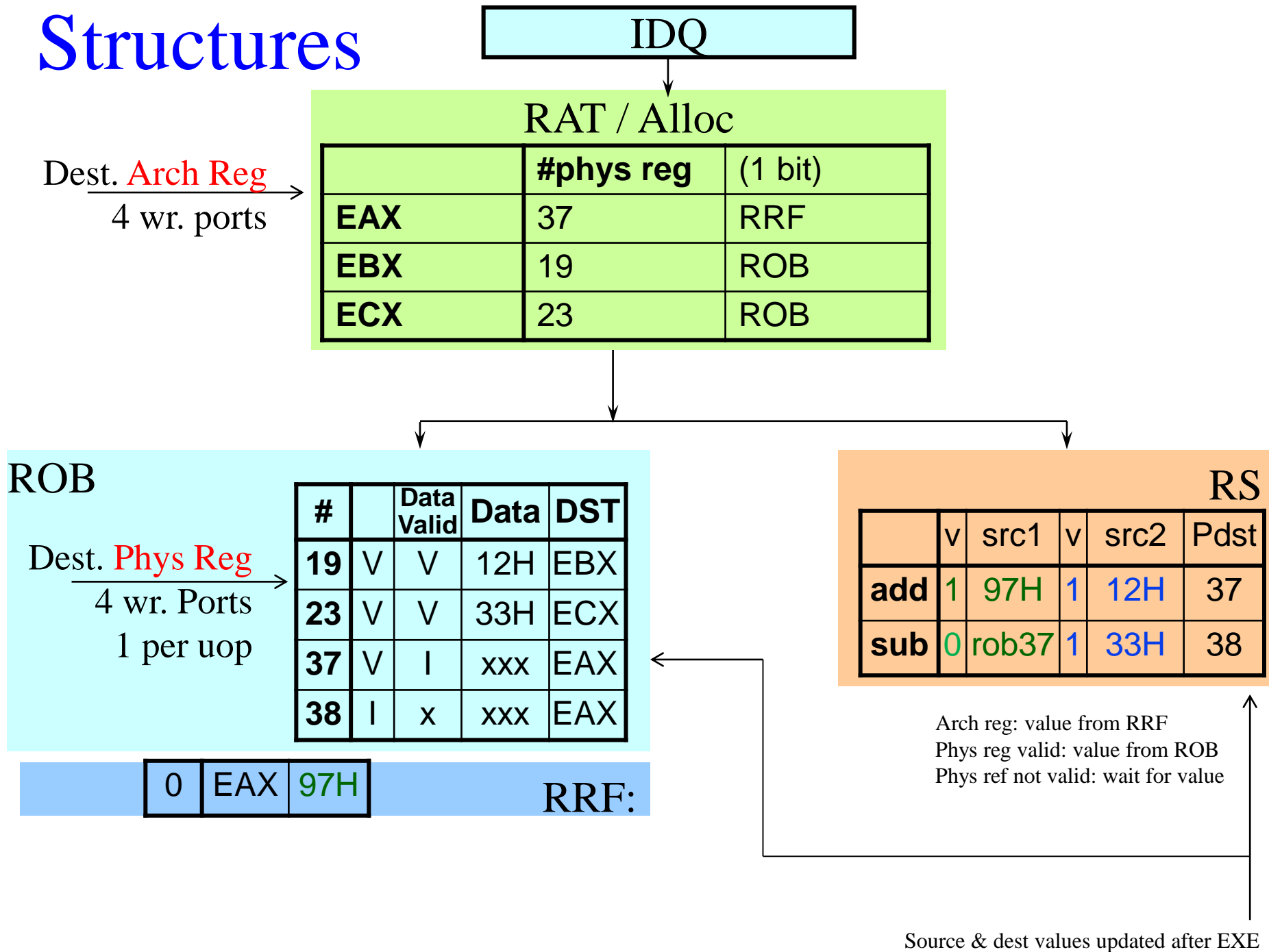
כעת אין תלויות מסוג
False Dependencies

אבל RAW עדיין קיים

ROB (reorder Buffer)

- דרך נפוצה למימוש הדבר היא ע"י ROB, שהוא חוצץ המקבל את ההוראות משלב ה-decode ע"פ הסדר, הכניסה התאימה ב-ROB מהווה את מספר הרגיסטר הזמני
- בשלב ה-EXE נכתבת התוצאה לכניסה המתאימה ב-ROB וההוראות מבצעות את שלב ה-commit ע"פ הסדר שלהן ב-ROB – הוראה יכולה לבצע commit רק אם זו שלפניה עשתה זאת.

Structures



P6

- כדוגמא ל-OOO ניקח את מכונת ה-P6 שממומשת במעבדי Pentium2 ו-Pentium3
- במכונה זו ההוראות מסוג CISC נכנסות לשלב ה-decode ע"פ הסדר
- שם הן מפורקות להוראות פנימיות הנקראות uops מסוג RISC
- הוראות אלו מתבצעות בשלב ה-out-of-order EXE
- לבסוף בשלב ה-commit (retire) ה-uops מחוברים בחזרה להוראות ה-CISC ע"פ הסדר המקורי של ההוראות
- המכונה גם משתמשת ב-register renaming הממומש ע"י ROB

דוגמא

DIV R2,R4,R3

LD R3,R4(50)

DIV R1,R2,R3

ADD R2,R4,R3

SUB R3,R2,R3

נויח כי:

Div: 4cc

Add/Sub: 1cc

Mem: 2cc

RAT

Arch
Reg.



R1	RF1
R2	RF2
R3	RF2
R4	RF3

$R3 \leftarrow R2 - R3$
$R2 \leftarrow R4 + R3$
$R1 \leftarrow R2 / R3$
$R3 \leftarrow \text{MEM}(R4 + 50)$
$R2 \leftarrow R4 / R3$

Instruction Q

MOB

ROB

Phys.
Reg.

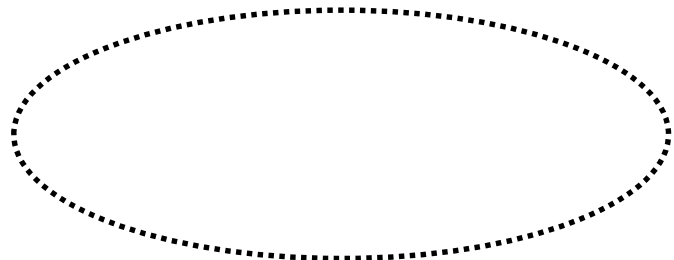


#	Val id	Data Valid	Data	DST
		X	XXX	X
		X	XXX	X
		X	XXX	X
		X	XXX	X
		X	XXX	X
		X	XXX	X
		X	XXX	X
		X	XXX	X

RS

RS example

	v	src1	v	src2	Pdst
add	1	97H	1	12H	37
sub	0	rob37	1	33H	38



Execute



Retire

RAT

Arch
Reg.



R1	RF0
R2	RF1
R3	RF2
R4	RF3



$R3 \leftarrow R2 - R3$
$R2 \leftarrow R4 + R3$
$R1 \leftarrow R2 / R3$
$R3 \leftarrow \text{MEM}(R4 + 50)$
$R2 \leftarrow R4 / R3$

Instruction Q

ROB

Phys.
Reg.



#	Val id	Data Valid	Data	DST
	0	x	xxx	X
	0			
	0			
	0			
	0			
	0			
	0			
	0			

RS

MOB



Execute



Retire

RAT

Arch
Reg.



R1	RF0
R2	RB0
R3	RF2
R4	RF3



$R3 \leftarrow R2 - R3$
$R2 \leftarrow R4 + R3$
$R1 \leftarrow R2 / R3$
$R3 \leftarrow \text{MEM}(R4 + 50)$
$R2 \leftarrow R4 / R3$

Instruction Q

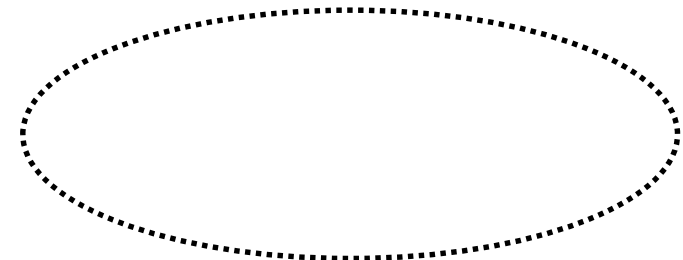
ROB

Comment	#	Valid	Data Valid	Data	DST
$R2 \leftarrow R4 / R3$	0	1	0	xxx	R2
	1	0			
	2	0			
	3	0			
	4	0			
	5	0			
	6	0			
	7	0			

RS

$RB0 \leftarrow R4 / R3$

MOB



Execute



Retire

RAT

Arch
Reg.



R1	RF1
R2	RB0
R3	RB1
R4	RF3



$R3 \leftarrow R2 - R3$
$R2 \leftarrow R4 + R3$
$R1 \leftarrow R2 / R3$
$R3 \leftarrow \text{MEM}(R4 + 50)$
$R2 \leftarrow R4 / R3$

Instruction Q

ROB

Comment	#	Valid	Data Valid	Data	DST
$R2 \leftarrow R4 / R3$	0	1	0	xxx	R2
$R3 \leftarrow \text{MEM}(r4 + 50)$	1	1	0	xxx	R3
	2	0			
	3	0			
	4	0			
	5	0			
	6	0			
	7	0			

RS

$RB0 \leftarrow R4 / R3$
$Mob \leftarrow R4 + 50$

MOB

$RB1 \leftarrow \text{MEM}(\text{not rdy})$




Execute



Retire

Arch
Reg.



RAT	
R1	RB2
R2	RB0
R3	RB1
R4	RF3



$R3 \leftarrow R2 - R3$
$R2 \leftarrow R4 + R3$
$R1 \leftarrow R2 / R3$
$R3 \leftarrow \text{MEM}(R4 + 50)$
$R2 \leftarrow R4 / R3$

Instruction Q

ROB					
Comment	#	Valid	Data Valid	Data	DST
$R2 \leftarrow R4 / R3$	0	1	0	xxx	R2
$R3 \leftarrow \text{MEM}(r4 + 50)$	1	1	0	xxx	R3
$R1 \leftarrow R2 / R3$	2	1	0	xxx	R1
	3	0			
	4	0			
	5	0			
	6	0			
	7	0			

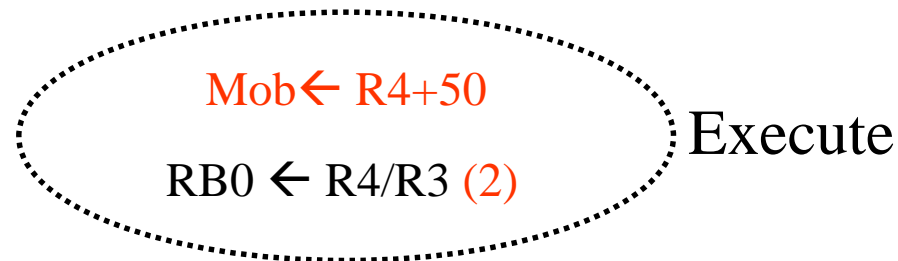
RS

$RB0 \leftarrow R4 / R3$
$\text{Mob} \leftarrow R4 + 50$
$RB2 \leftarrow \underline{RB0} / \underline{RB1}$

MOB

$RB1 \leftarrow \text{MEM}(\text{not Rdy})$

W



RAT

Arch
Reg. →

R1	RB2
R2	RB3
R3	RB1
R4	RF3



$R3 \leftarrow R2 - R3$
$R2 \leftarrow R4 + R3$
$R1 \leftarrow R2 / R3$
$R3 \leftarrow \text{MEM}(R4 + 50)$
$R2 \leftarrow R4 / R3$

Instruction Q

ROB

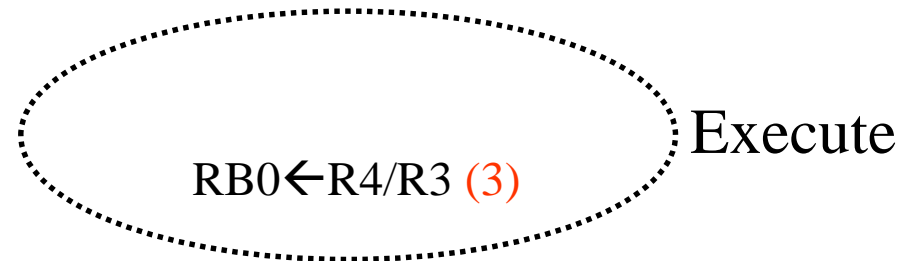
Comment	#	Valid	Data Valid	Data	DST
$R2 \leftarrow R4 / R3$	0	1	0	xxx	R2
$R3 \leftarrow \text{MEM}(r4 + 50)$	1	1	0	xxx	R3
$R1 \leftarrow R2 / R3$	2	1	0	xxx	R1
$R2 \leftarrow R4 + R3$	3	1	0	xxx	R2
	4	0			
	5	0			
	6	0			
	7	0			

RS

$RB0 \leftarrow R4 / R3$
$RB3 \leftarrow R4 + \underline{RB1}$
$RB2 \leftarrow \underline{RB0} / \underline{RB1}$ W

MOB

$RB1 \leftarrow \text{MEM}(R4 + 50)$ (1)



RAT

R1	RB2
R2	RB3
R3	RB4
R4	RF3

Arch
Reg.

$R3 \leftarrow R2 - R3$
$R2 \leftarrow R4 + R3$
$R1 \leftarrow R2 / R3$
$R3 \leftarrow \text{MEM}(R4 + 50)$
$R2 \leftarrow R4 / R3$

Instruction Q

ROB

Comment	#	Valid	Data Valid	Data	DST
$R2 \leftarrow R4 / R3$	0	1	0	xxx	R2
$R3 \leftarrow \text{MEM}(r4 + 50)$	1	1	0	xxx	R3
$R1 \leftarrow R2 / R3$	2	1	0	xxx	R1
$R2 \leftarrow R4 + R3$	3	1	0	xxx	R2
$R3 \leftarrow R2 - R3$	4	1	0	xxx	R3
	5	0			
	6	0			
	7	0			

RS

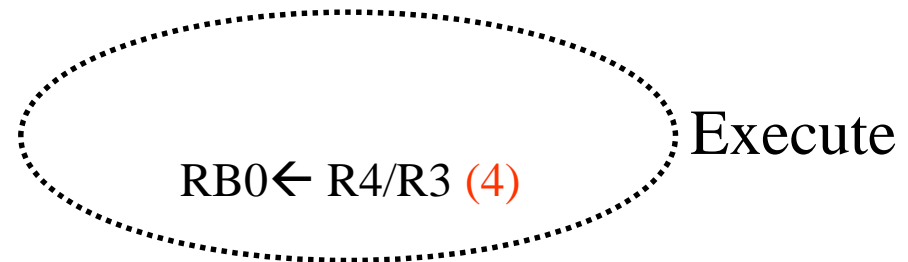
$RB0 \leftarrow R4 / R3$
$RB3 \leftarrow R4 + \underline{RB1}$
$RB2 \leftarrow \underline{RB0} / \underline{RB1}$
$RB4 \leftarrow \underline{RB3} - \underline{RB1}$

W

W

MOB

$RB1 \leftarrow \text{MEM}(R4 + 50)$ (2)



RAT

R1	RB2
R2	RB3
R3	RB4
R4	RF3

Arch
Reg.

$R3 \leftarrow R2 - R3$
$R2 \leftarrow R4 + R3$
$R1 \leftarrow R2 / R3$
$R3 \leftarrow \text{MEM}(R4 + 50)$
$R2 \leftarrow R4 / R3$

Instruction Q

ROB

Comment	#	Valid	Data Valid	Data	DST
$R2 \leftarrow R4 / R3$	0	1	1	Value2	R2
$R3 \leftarrow \text{MEM}(r4 + 50)$	1	1	1	Value3	R3
$R1 \leftarrow R2 / R3$	2	1	0	xxx	R1
$R2 \leftarrow R4 + R3$	3	1	0	xxx	R2
$R3 \leftarrow R2 - R3$	4	1	0	xxx	R3
	5	0			
	6	0			
	7	0			

RS

$RB3 \leftarrow R4 + RB1$
$RB2 \leftarrow RB0 / RB1$
$RB4 \leftarrow \underline{RB3} - RB1$

W

MOB

$RB3 \leftarrow R4 + RB1$
 $RB2 \leftarrow RB0 / RB1$ (1)

Execute

Retire

RAT

Arch
Reg.



R1	RB2
R2	RB3
R3	RB4
R4	RF3

$R3 \leftarrow R2 - R3$
$R2 \leftarrow R4 + R3$
$R1 \leftarrow R2 / R3$
$R3 \leftarrow \text{MEM}(R4 + 50)$
$R2 \leftarrow R4 / R3$

Instruction Q

ROB

Comment	#	Valid	Data Valid	Data	DST
$R2 \leftarrow R4 / R3$	0	1	1	Value2	R2
$R3 \leftarrow \text{MEM}(r4 + 50)$	1	1	1	Value3	R3
$R1 \leftarrow R2 / R3$	2	1	0	xxx	R1
$R2 \leftarrow R4 + R3$	3	1	1	Value4	R2
$R3 \leftarrow R2 - R3$	4	1	0	xxx	R3
	5	0			
	6	0			
	7	0			

RS

$RB2 \leftarrow RB0 / RB1$
$RB4 \leftarrow RB3 - RB1$

MOB



Execute



Retire

RAT

Arch
Reg.



R1	RB2
R2	RB3
R3	RB4
R4	RF3

$R3 \leftarrow R2 - R3$
$R2 \leftarrow R4 + R3$
$R1 \leftarrow R2 / R3$
$R3 \leftarrow \text{MEM}(R4 + 50)$
$R2 \leftarrow R4 / R3$

Instruction Q

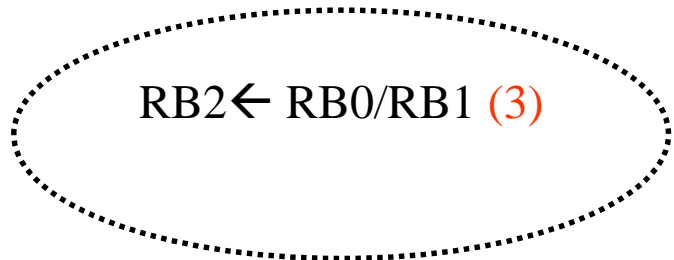
ROB

Comment	#	Valid	Data Valid	Data	DST
$R2 \leftarrow R4 / R3$	0	0			
$R3 \leftarrow \text{MEM}(r4 + 50)$	1	1	1	Value3	R3
$R1 \leftarrow R2 / R3$	2	1	0	xxx	R1
$R2 \leftarrow R4 + R3$	3	1	1	Value4	R2
$R3 \leftarrow R2 - R3$	4	1	1	Value5	R3
	5	0			
	6	0			
	7	0			

RS

$RB2 \leftarrow RB0 / RB1$

MOB



Execute



Retire

RAT

R1	RB2
R2	RB3
R3	RB4
R4	RF3

Arch
Reg.

$R3 \leftarrow R2 - R3$
$R2 \leftarrow R4 + R3$
$R1 \leftarrow R2 / R3$
$R3 \leftarrow \text{MEM}(R4 + 50)$
$R2 \leftarrow R4 / R3$

Instruction Q

ROB

Comment	#	Valid	Data Valid	Data	DST
$R2 \leftarrow R4 / R3$	0	0			
$R3 \leftarrow \text{MEM}(r4 + 50)$	1	0			
$R1 \leftarrow R2 / R3$	2	1	0	xxx	R1
$R2 \leftarrow R4 + R3$	3	1	1	Value4	R2
$R3 \leftarrow R2 - R3$	4	1	1	Value5	R3
	5	0			
	6	0			
	7	0			

RS

$RB2 \leftarrow RB0 / RB1$

MOB

 $RB2 \leftarrow RB0 / RB1$ (4)

Execute

Retire

RAT

R1	RB2
R2	RB3
R3	RB4
R4	RF3

Arch
Reg.

$R3 \leftarrow R2 - R3$
$R2 \leftarrow R4 + R3$
$R1 \leftarrow R2 / R3$
$R3 \leftarrow \text{MEM}(R4 + 50)$
$R2 \leftarrow R4 / R3$

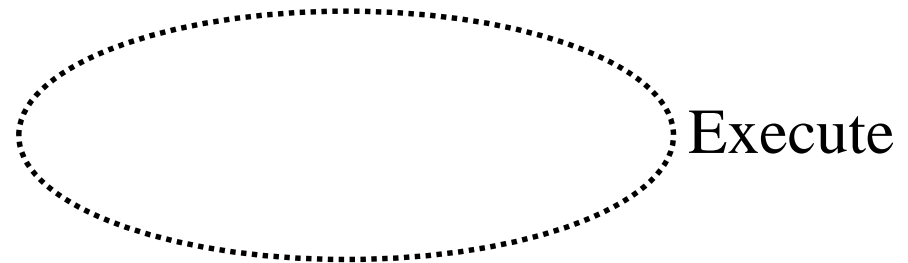
Instruction Q

ROB

Comment	#	Valid	Data Valid	Data	DST
$R2 \leftarrow R4 / R3$	0	0			
$R3 \leftarrow \text{MEM}(r4 + 50)$	1	0			
$R1 \leftarrow R2 / R3$	2	1	1	Value6	R1
$R2 \leftarrow R4 + R3$	3	1	1	Value4	R2
$R3 \leftarrow R2 - R3$	4	1	1	Value5	R3
	5	0			
	6	0			
	7	0			

RS

MOB



Execute



Retire

RAT

Arch
Reg.



R1	
R2	
R3	
R4	RF3

$R3 \leftarrow R2 - R3$
$R2 \leftarrow R4 + R3$
$R1 \leftarrow R2 / R3$
$R3 \leftarrow \text{MEM}(R4 + 50)$
$R2 \leftarrow R4 / R3$

Instruction Q

ROB

Comment	#	Valid	Data Valid	Data	DST
$R2 \leftarrow R4 / R3$	0	0			
$R3 \leftarrow \text{MEM}(r4 + 50)$	1	0			
$R1 \leftarrow R2 / R3$	2	1	1	Value6	R1
$R2 \leftarrow R4 + R3$	3	1	1	Value4	R2
$R3 \leftarrow R2 - R3$	4	1	1	Value5	R3
	5	0			
	6	0			
	7	0			

RS

MOB



Execute

$R1 \leftarrow RB2$
$R2 \leftarrow RB3$
$R3 \leftarrow RB4$

Retire

RAT

Arch
Reg.



R1	RF1
R2	RF2
R3	RF2
R4	RF3

$R3 \leftarrow R2 - R3$
$R2 \leftarrow R4 + R3$
$R1 \leftarrow R2 / R3$
$R3 \leftarrow \text{MEM}(R4 + 50)$
$R2 \leftarrow R4 / R3$

Instruction Q

ROB

Comment		Valid	Data Valid	Data	DST
$R2 \leftarrow R4 / R3$		0			
$R3 \leftarrow \text{MEM}(r4 + 50)$		0			
$R1 \leftarrow R2 / R3$		0			
$R2 \leftarrow R4 + R3$		0			
$R3 \leftarrow R2 - R3$		0			
		0			
		0			
		0			

RS

MOB



Execute



Retire

Backup

כיצד ניתן לשפר את ביצועי המערכת

- ב-**pipeline** של MIPS אותו למדנו זה מכבר, הייתה יחידת ALU בודדת שדאגה לביצוע כל הפעולות (שלב ה-EXE)
- הבעיה: ישנן פעולות כבדות יותר וכבדות פחות – למשל חיבור שלמים יהיה קל יותר מפעולת חילוק. מכיוון שמחזור השעון אמור להספיק לביצוע כל פעולה הרי שבעקבות הפעולות הכבדות נקבל מחזור שעון ארוך – פגיעה בביצועים.

פתרון 1

- נאפשר רק ביצוע הוראות פשוטות שלוקחות זמן קצר, והוראות כבדות יפורקו למספר הוראות פשוטות (מעבדי ה-RISC הראשונים היו כאלה)

חסרונות:

✓ ישנן פעולות כגון חישובי **floating-point** אשר קשה לפרק אותם לחלקים בעלי זמן ביצוע שווה

✓ ישנן פעולות כגון גישה לזיכרון שזמן הביצוע (גישה) אינו קבוע (תלוי בעומס על ה-bus) או משתנה מדור לדור

2 פתרון

- נהפוך את שלב ה-EXE ל - pipeline (או ל-multicycle) כך הוראה כבדה תתבצע במספר שלבים ונוכל לשמור על מחזור שעון קטן

חסרונות:

- ✓ לא ברור שתמיד נוכל לפרק הוראה כבדה לשלבי pipeline
- ✓ הוספת שלבים מגדילה את ה-penalty שאנו משלמים על חיזויים שגויים וכן יוצרת בעיות עם תלויות המידע (data hazard) – אמנם נקטין את מחזור השעון אבל אנו עלולים לפגוע ב-CPI.

פתרון 3

- נבנה pipeline שונה לפקודות בעלות זמן ביצוע שונה

- נאפשר להוראות מ-pipeline שונה להסתיים שלא ע"פ סדר הופעתן בקוד

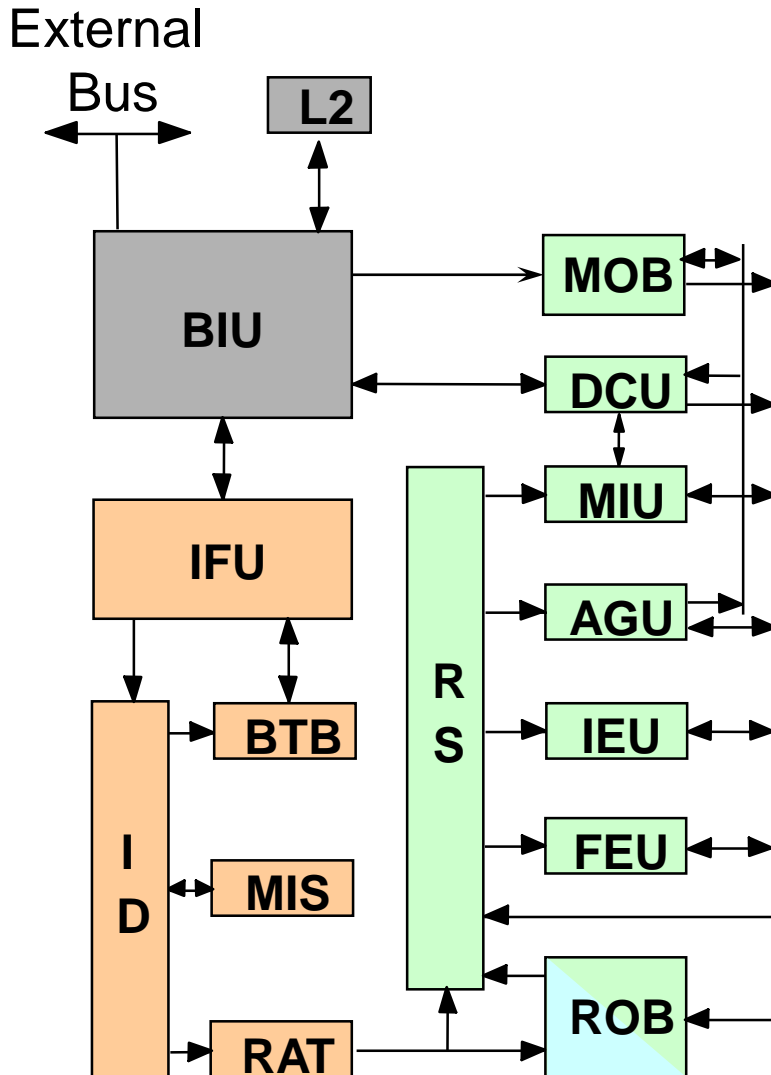
— קרי, כל עוד התוכנית שומרת על נכונותה, נאפשר לפקודה שמופיעה בקוד מאוחר יותר להסתיים לפני שפקודה שקדמה לה מסתיימת

- שיטה זאת מאפשרת ביצוע של מספר הוראות במקביל:

— כל עוד שייכות ל-pipeline שונה

— כל עוד אינן תלויות אחת בשניה. כך נוכל לשפר את ה-CPI של המכונה ובכך לשפר ביצועים.

OOOE – The P6 Example



- **In-Order Front End**

- BIU: Bus Interface Unit
- IFU: Instruction Fetch Unit (includes IC)
- BTB: Branch Target Buffer
- ID: Instruction Decoder
- MIS: Micro-Instruction Sequencer
- RAT: Register Alias Table

- **Out-of-order Core**

- ROB: Reorder Buffer
- RRF: Real Register File
- RS: Reservation Stations
- IEU: Integer Execution Unit
- FEU: Floating-point Execution Unit
- AGU: Address Generation Unit
- MIU: Memory Interface Unit
- DCU: Data Cache Unit
- MOB: Memory Order Buffer
- L2: Level 2 cache

- **In-Order Retire**