MIPS

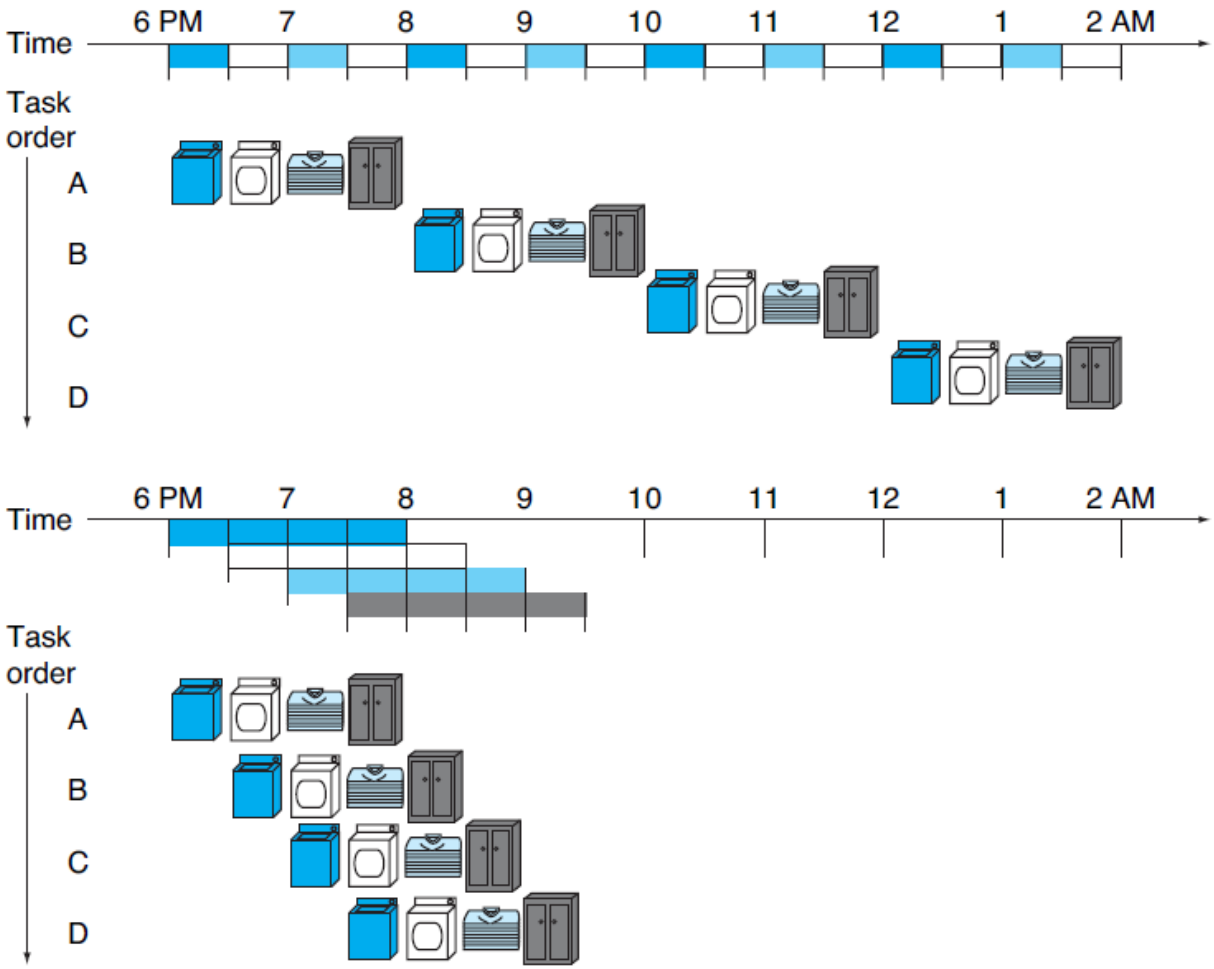
מימוש מצונר

מימוש Pipeline - מוטיבציה

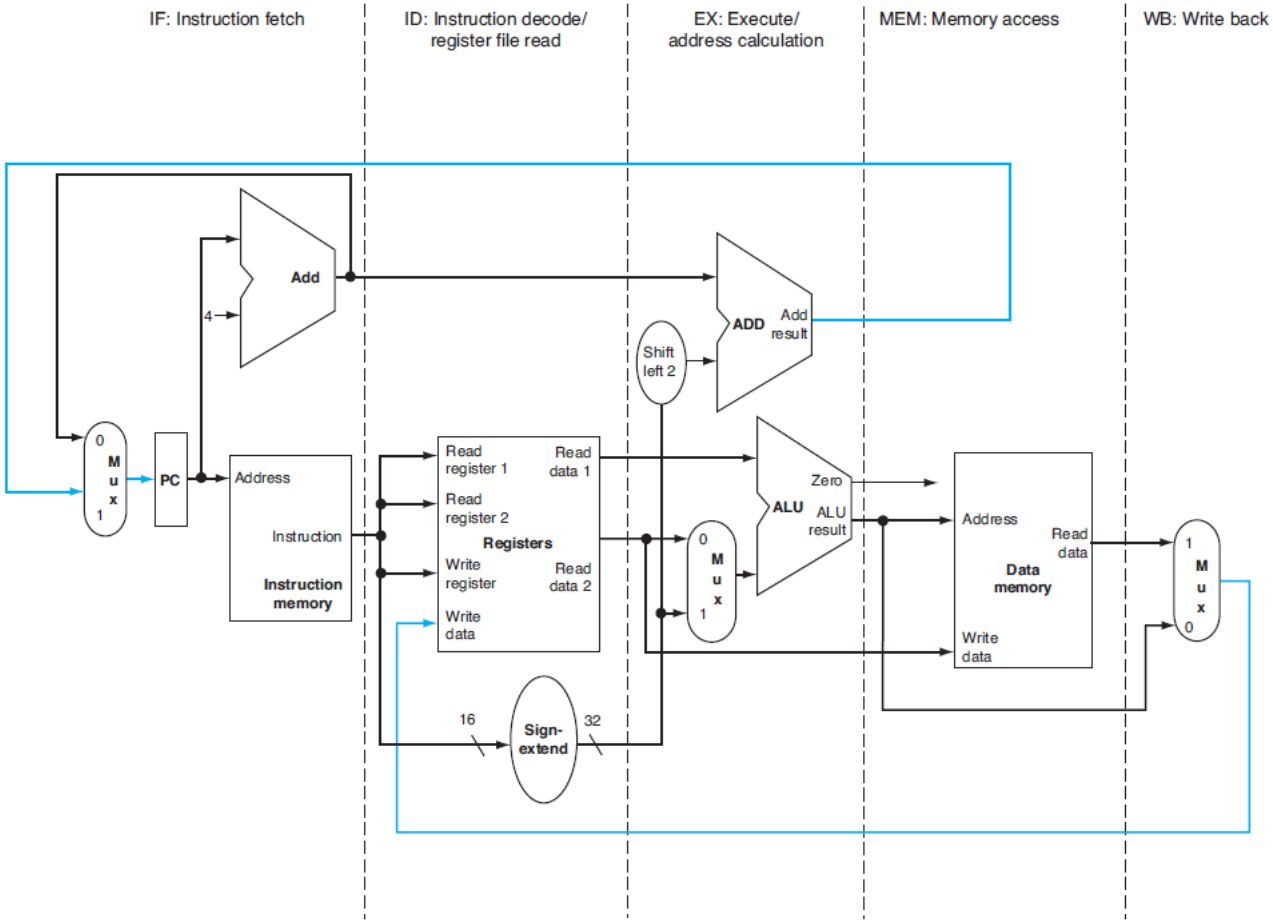
- במימושים הקודמים הפקודות התבצעו אחת אחרי השניה.
  - רק כאשר פקודה הסתיימה נשלפה הפקודה הבאה מהזיכרון.

- במימוש ה-Pipeline המעבד עוסק במספר פקודות בו זמנית, כל פקודה בשלב אחר של ביצוע
  - מימוש בצורה זו יאפשר לנו לעלות את הספיקה (throughput) של המערכת

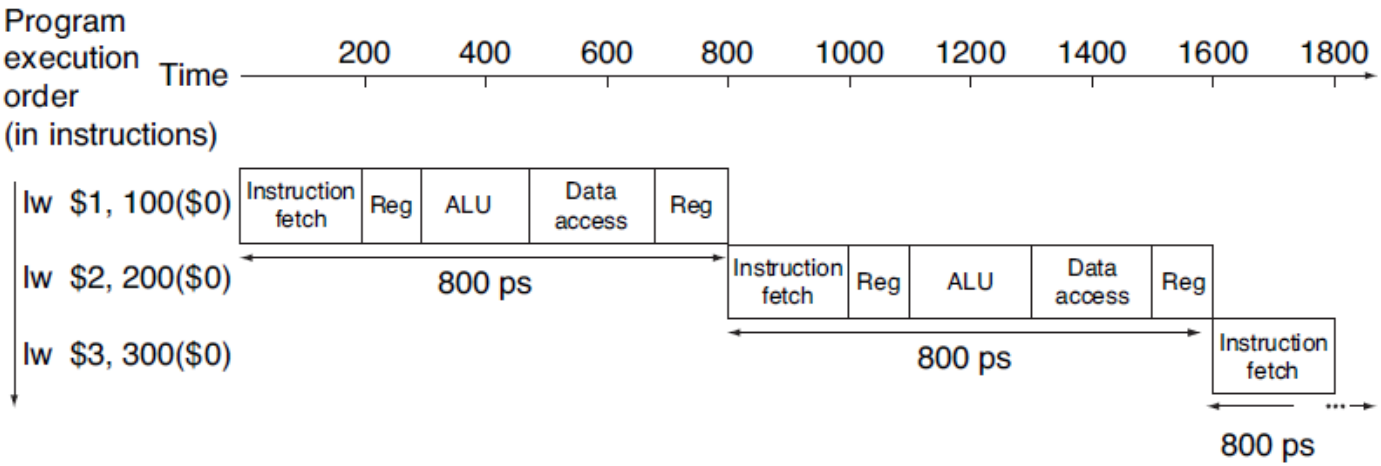
דוגמא לצינור

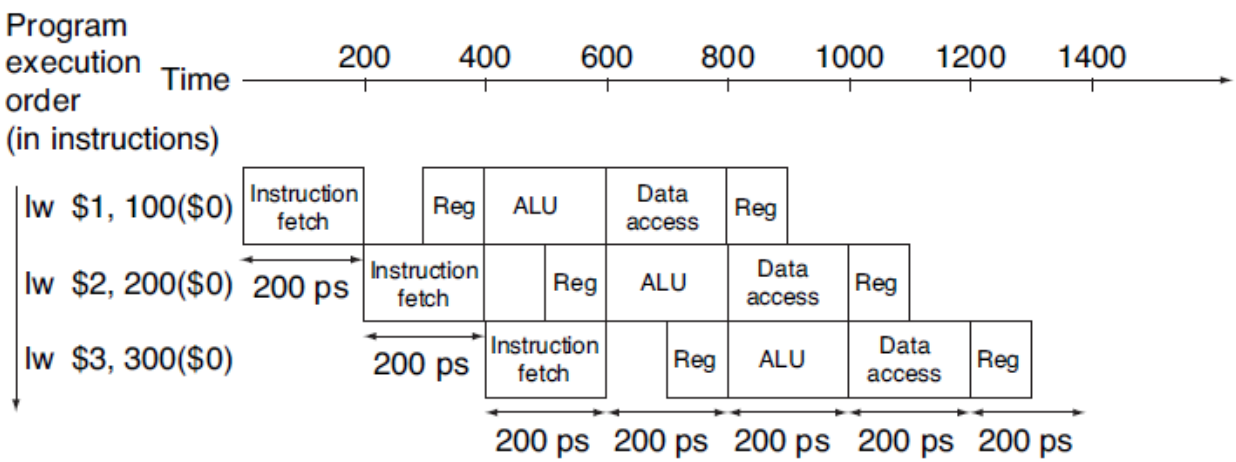


שלבי הביצוע במימוש Single-Cycle



עקרון מימוש הצינור במעבד ה-MIPS

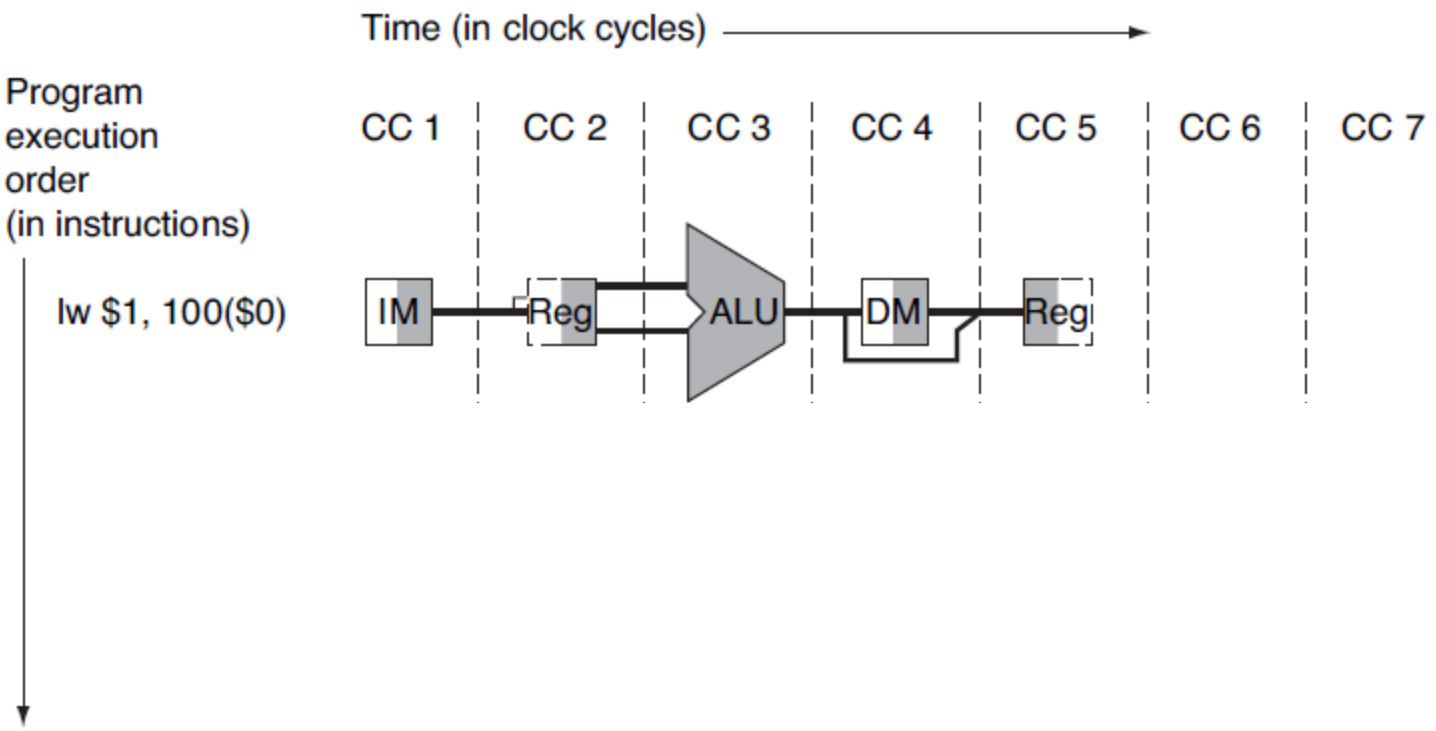




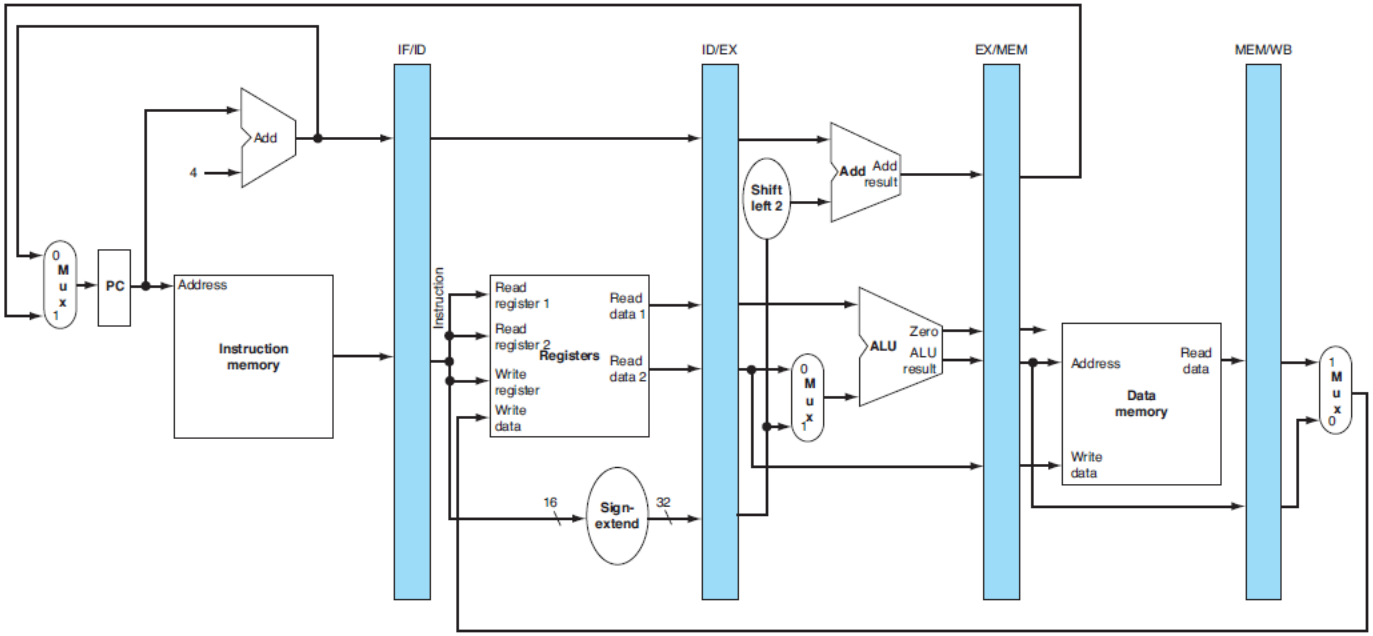
השוואת ביצועים

- ללא צינור:
  - זמן המחזור נקבע לפי הפקודה האיטית ביותר.
  - הכמיסות = מחזור שעון = 0.8ns
  - CPI=1
  - הספיקה = 1/0.8ns
- עם צינור:
  - זמן המחזור נקבע לפי השלב האיטי ביותר.
  - הכמיסות = עומק הצינור X זמן מחזור = 5x0.2ns = 1ns
  - ביצוע כל פקודה ימשך 5 מחזורי שעון (גם אם אפשר בפחות)
  - CPI = 1 כל מחזור שעון מסתיימת פקודה (מצב אידיאלי)
  - הספיקה (האידיאלית) = 1/0.2ns

ביצוע פקודות lw בצינור

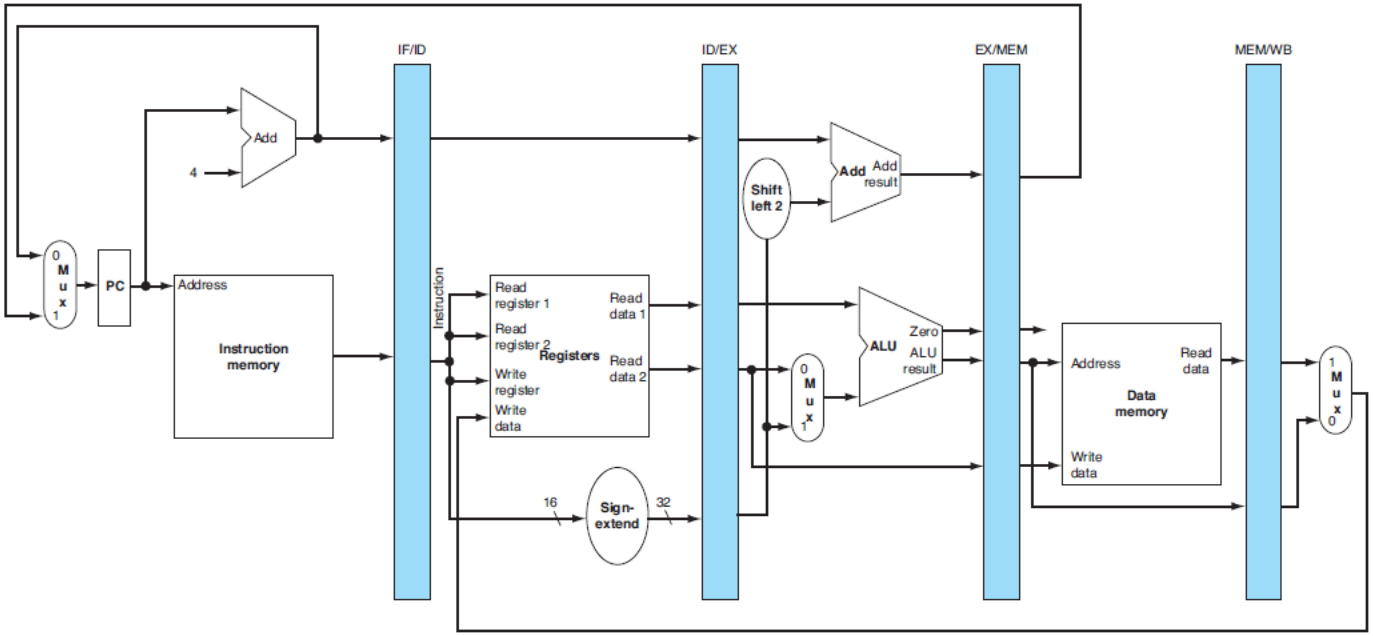


הוספת רגיסטרים לצינור מסלול הנתונים



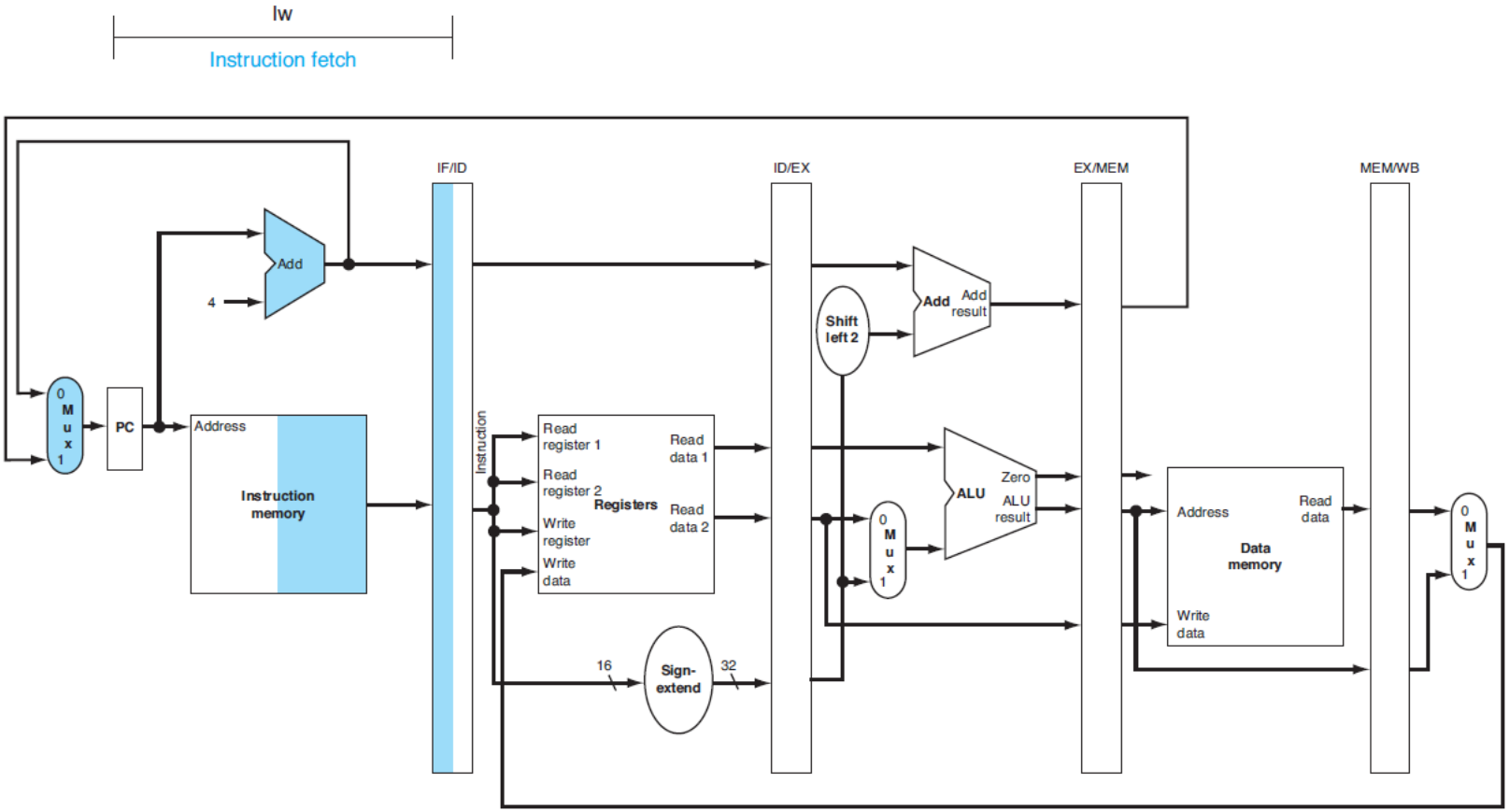
- הרגיסטרים צריכים להכיל את כל המידע שעובר בין השלבים
  - לדוגמא, רגיסטר IF/ID צריך להיות ברוחב 64 ביט

הוספת רגיסטרים לצינור מסלול הנתונים

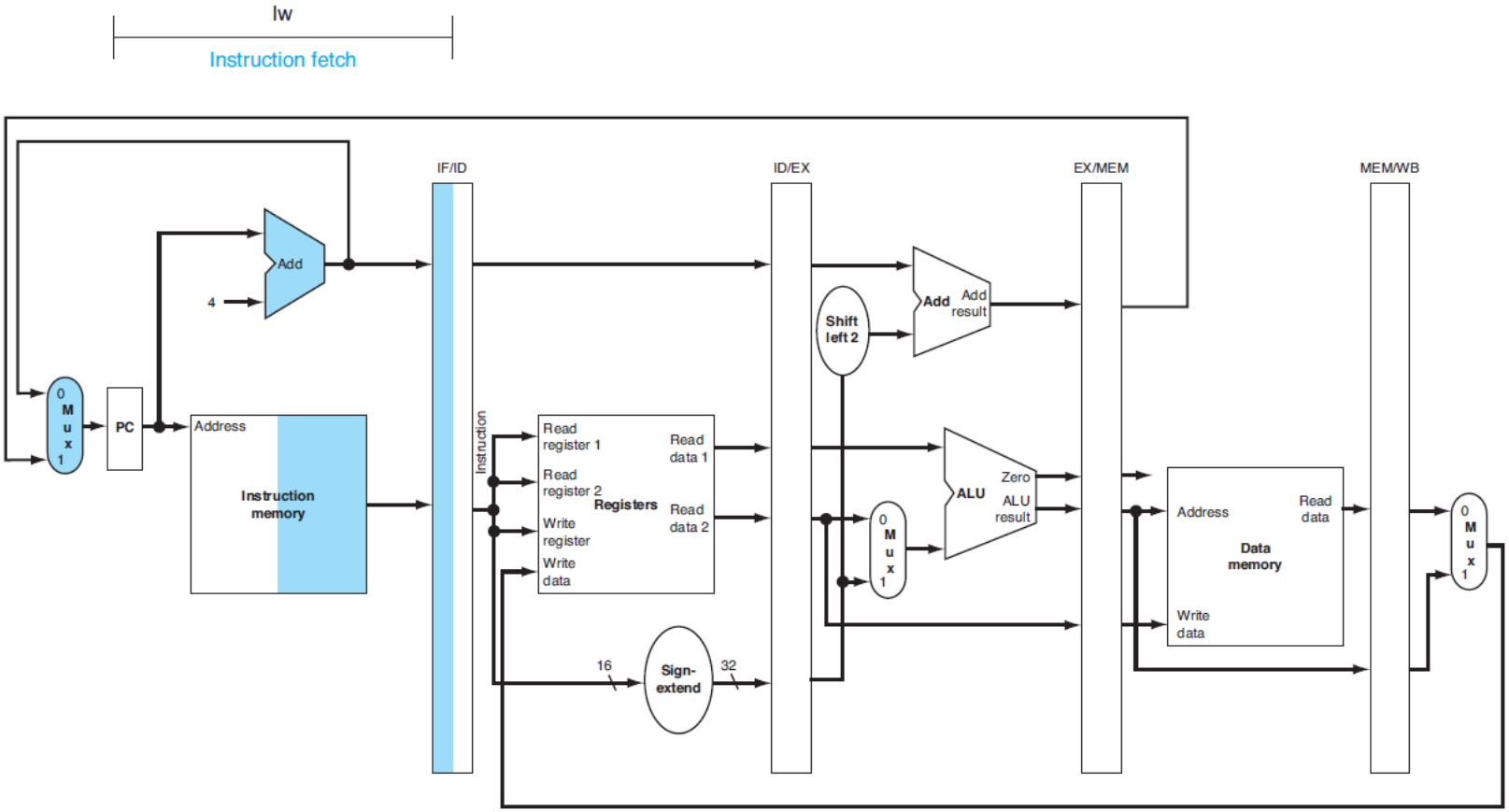


- הצינור מכיל 5 שלבים אך מספיקים 4 רגיסטרים להפרדה בין השלבים
  - השלב האחרון תמיד מסתיים בכתיבה לרגיסטר

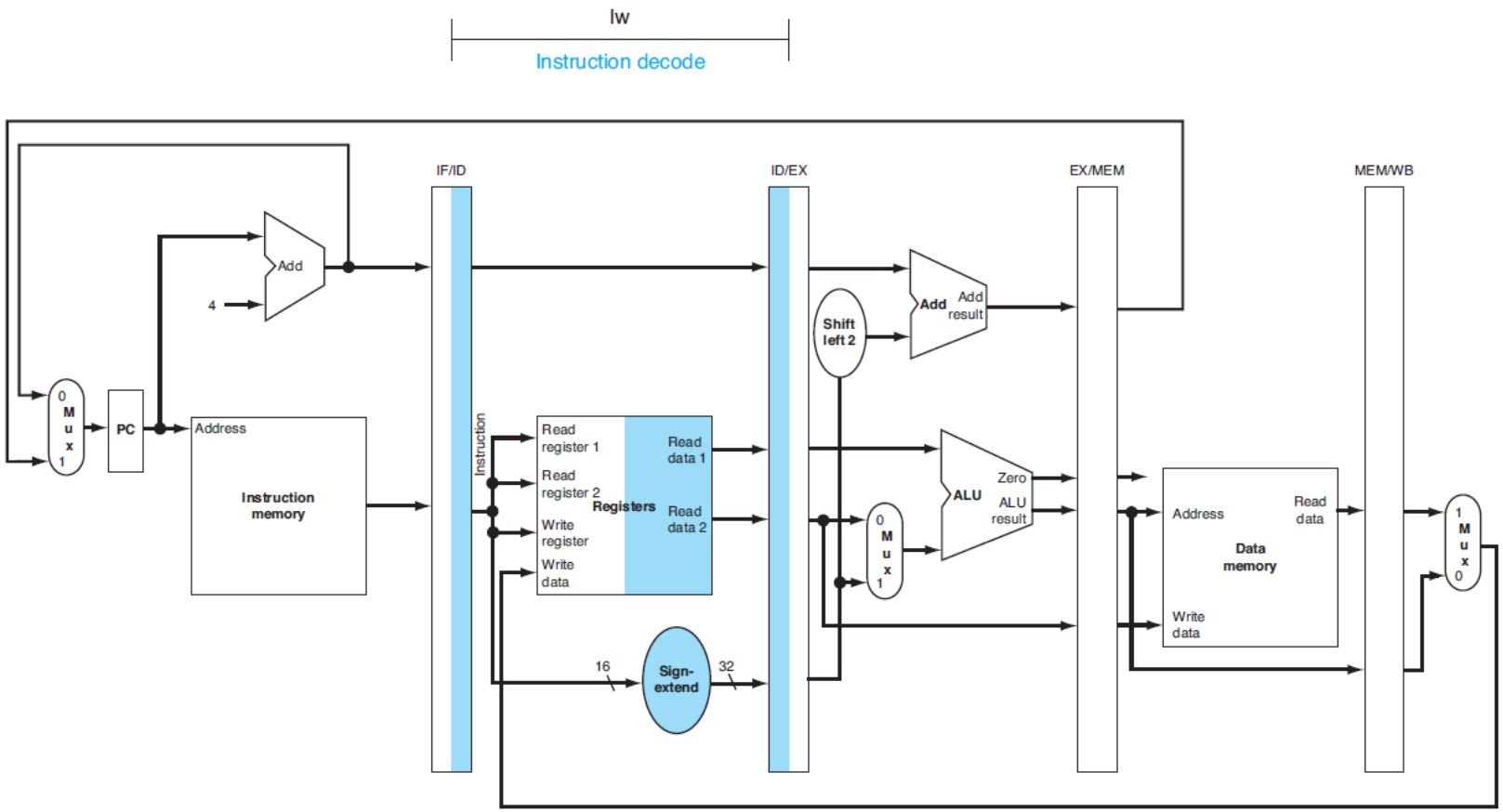
דוגמא לביצע פקודת lw



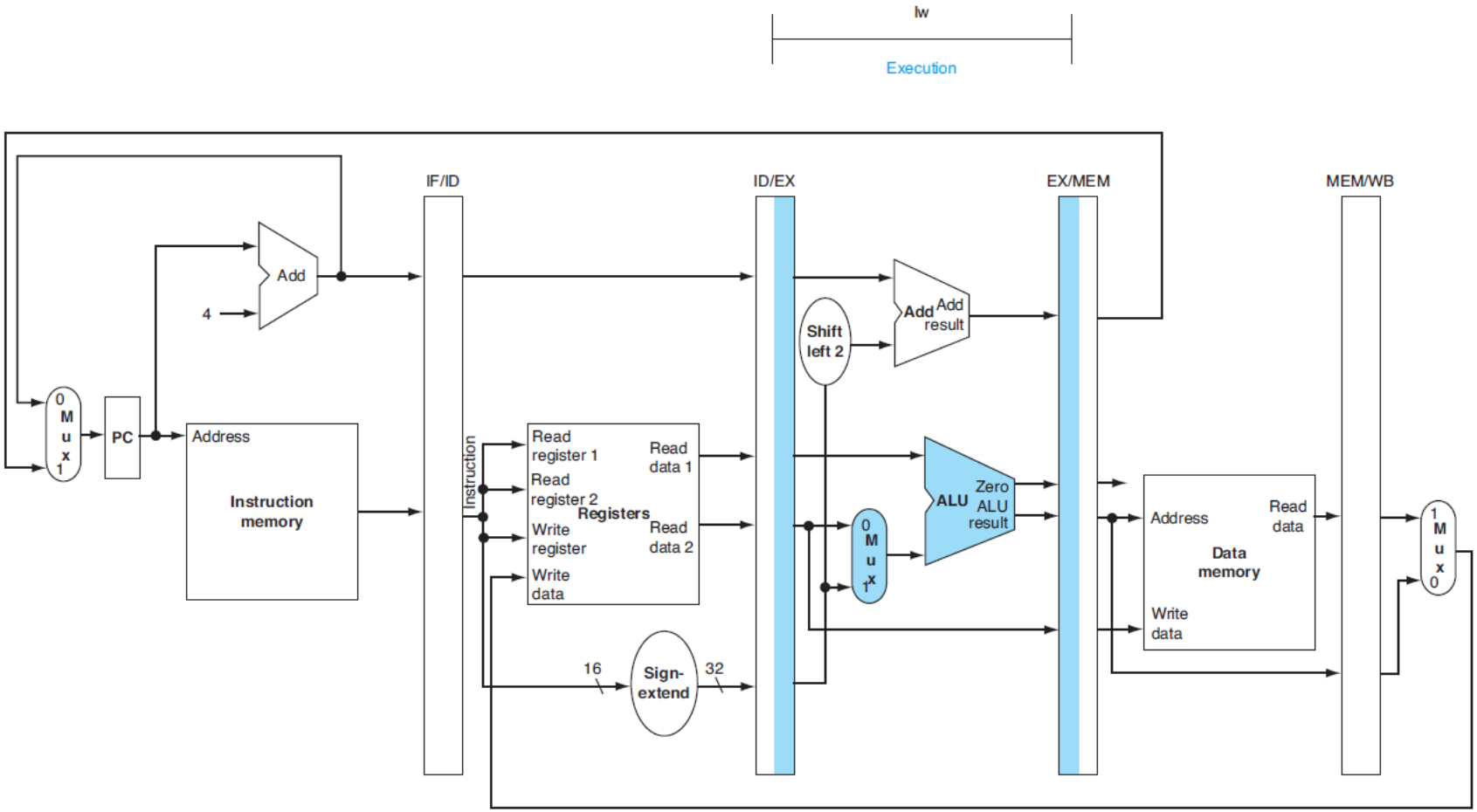
דוגמא לביצע פקודת lw



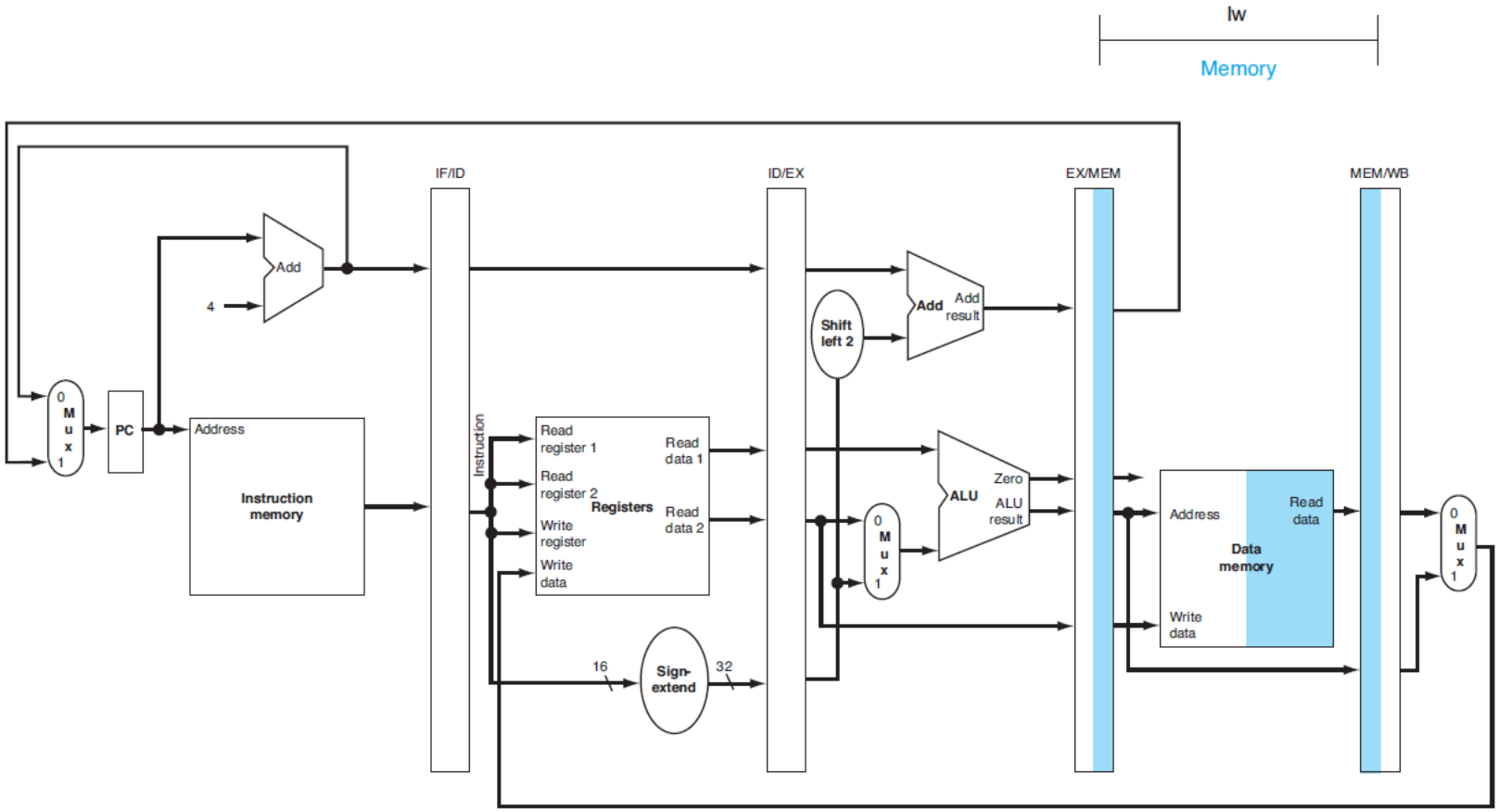
דוגמא לביצע פקודת lw



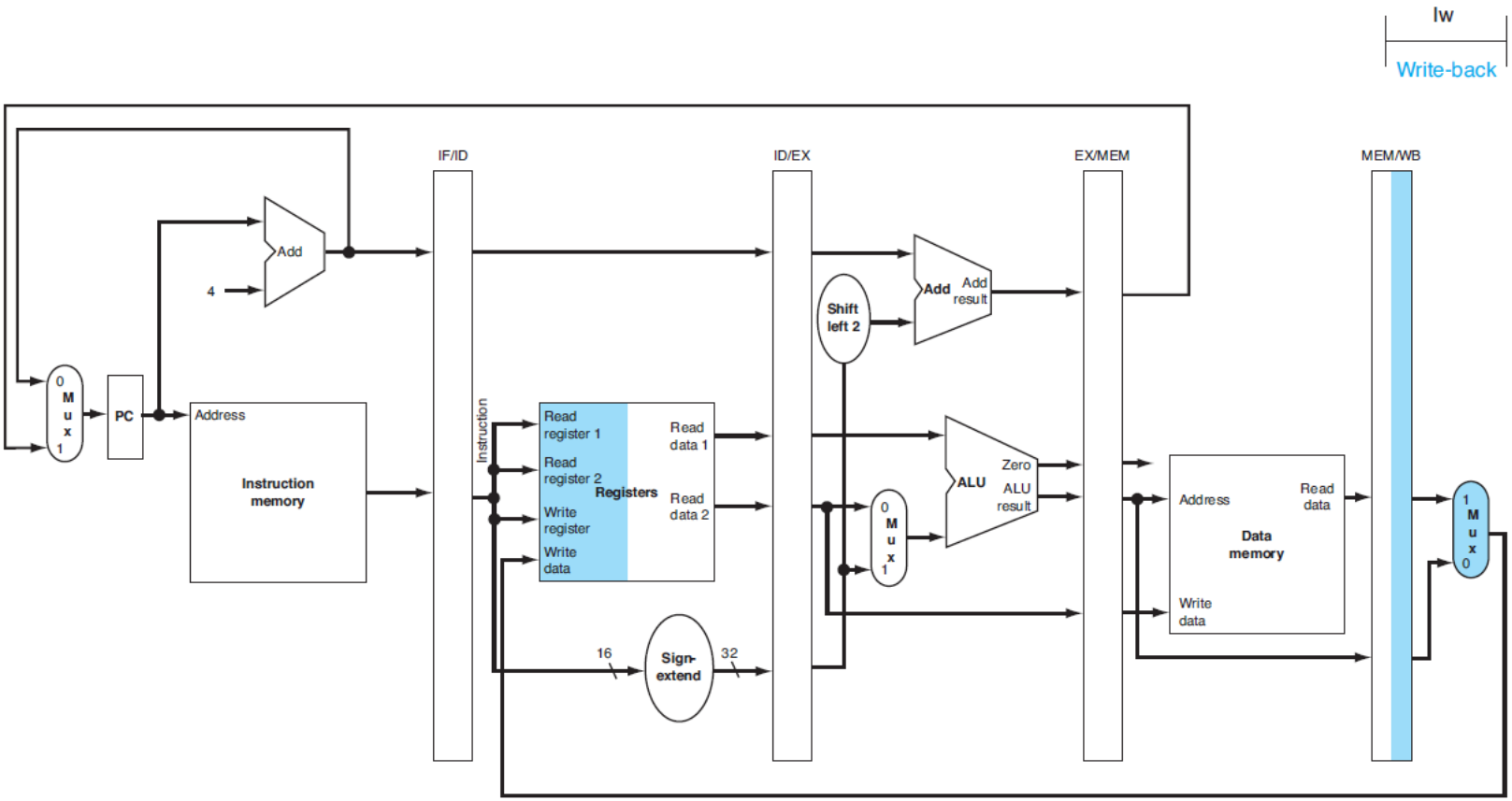
דוגמא לביצע פקודת lw



דוגמא לביצע פקודת lw



דוגמא לביצע פקודת lw



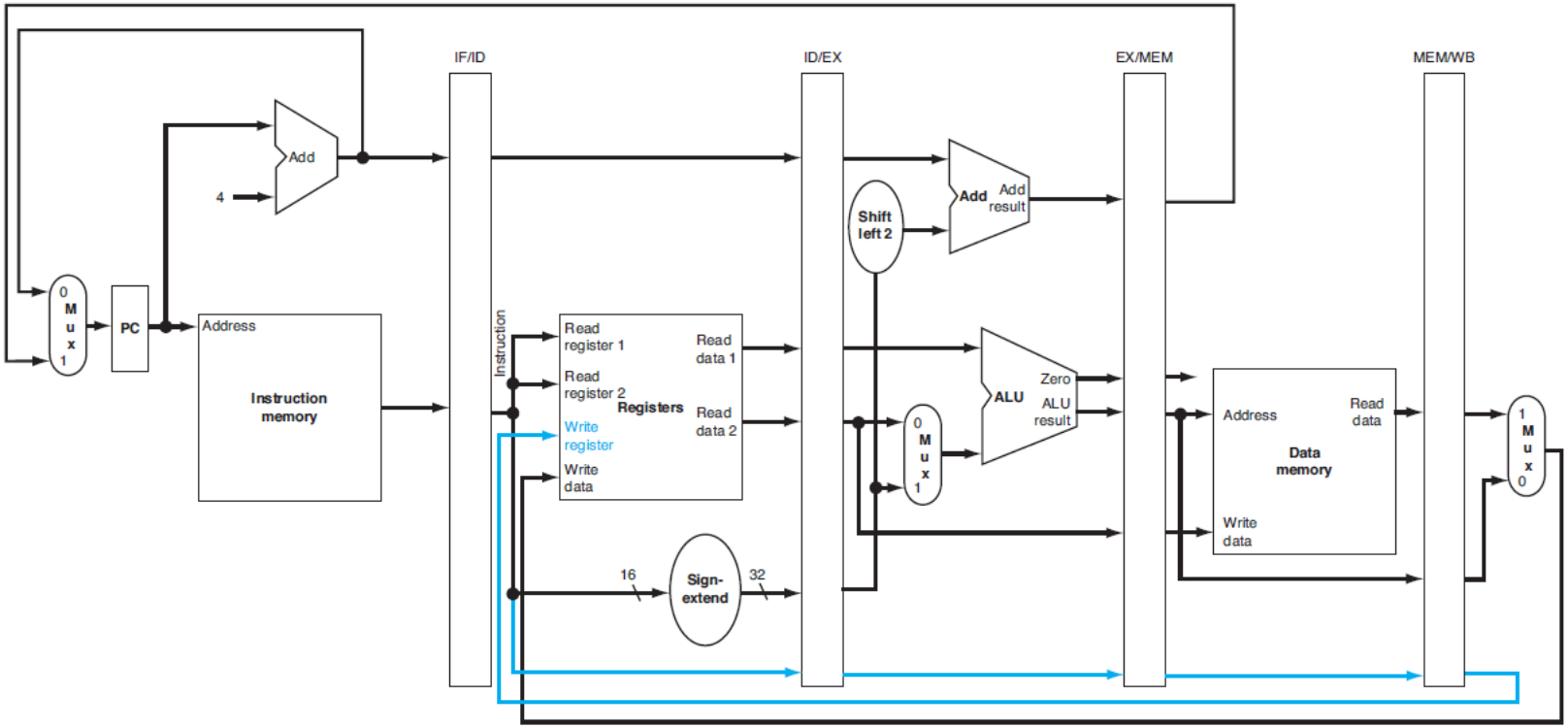
העברת נתונים במסלול הנתונים המצונר

- במקרה שצריכים נתונים משלב כלשהו בשלב מאוחר בצינור, חייבים להעביר נתונים אלו לאורך הצינור.

- בפקודת lw כפי שמומשה בשקפים הקודמים יש בעיה, מהי? כיצד ניתן לפתור אותה?

- עד שמגיעים לשלב ה-write–back הרגיסטר בו יש לאחסן את הנתונים (rt) שייך לפקודה אחרת.
- הפתרון – להעביר את מספר הרגיסטר בצינור, יחד עם הנתונים.

העברת נתונים במסלול הנתונים המצונר



צורת הפתרון תבוא לידי ביטוי גם עם אותות אחרים במעבד

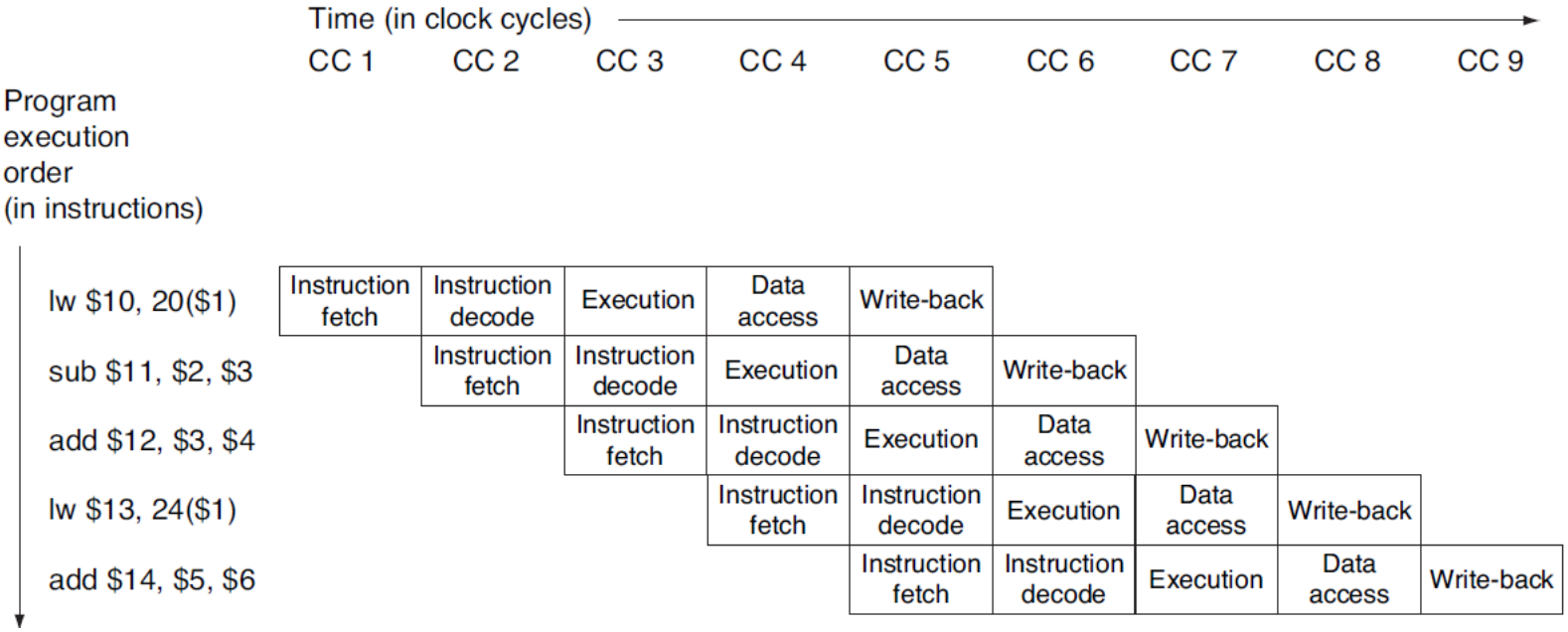
דוגמא

- נתונה סדרת הפקודות הבאה:

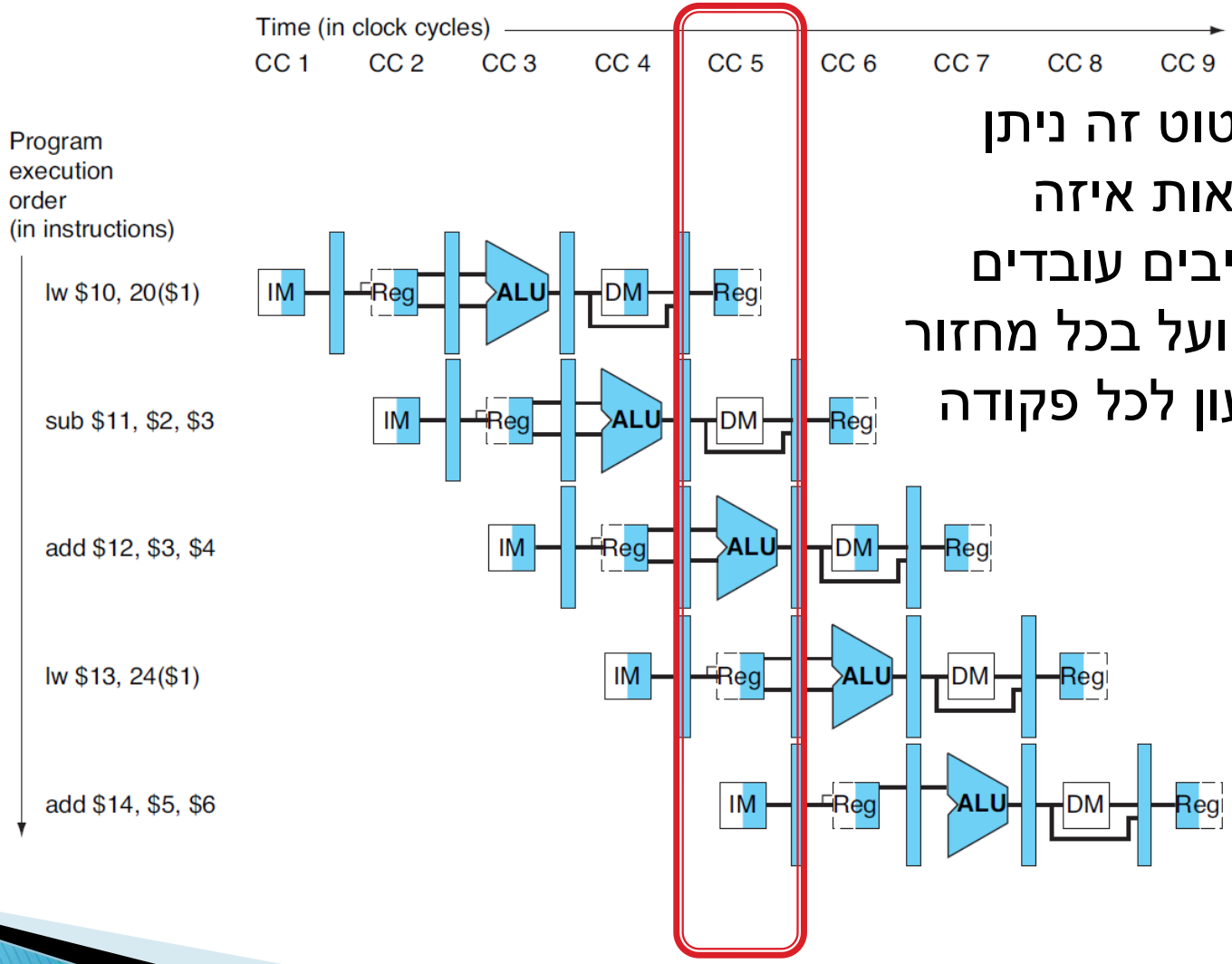
```
lw $10, 20($1)
sub $11, $2, $3
add $12, $3, $4
lw $13, 24($1)
add $14, $5, $6
```

- יש להציג את שלבי ביצוע הפקודות:
  - מה מבצעת כל יחידה במעבד בכל מחזור שעון?

דיאגרמה קלאסית להצגת הרצה בצינור



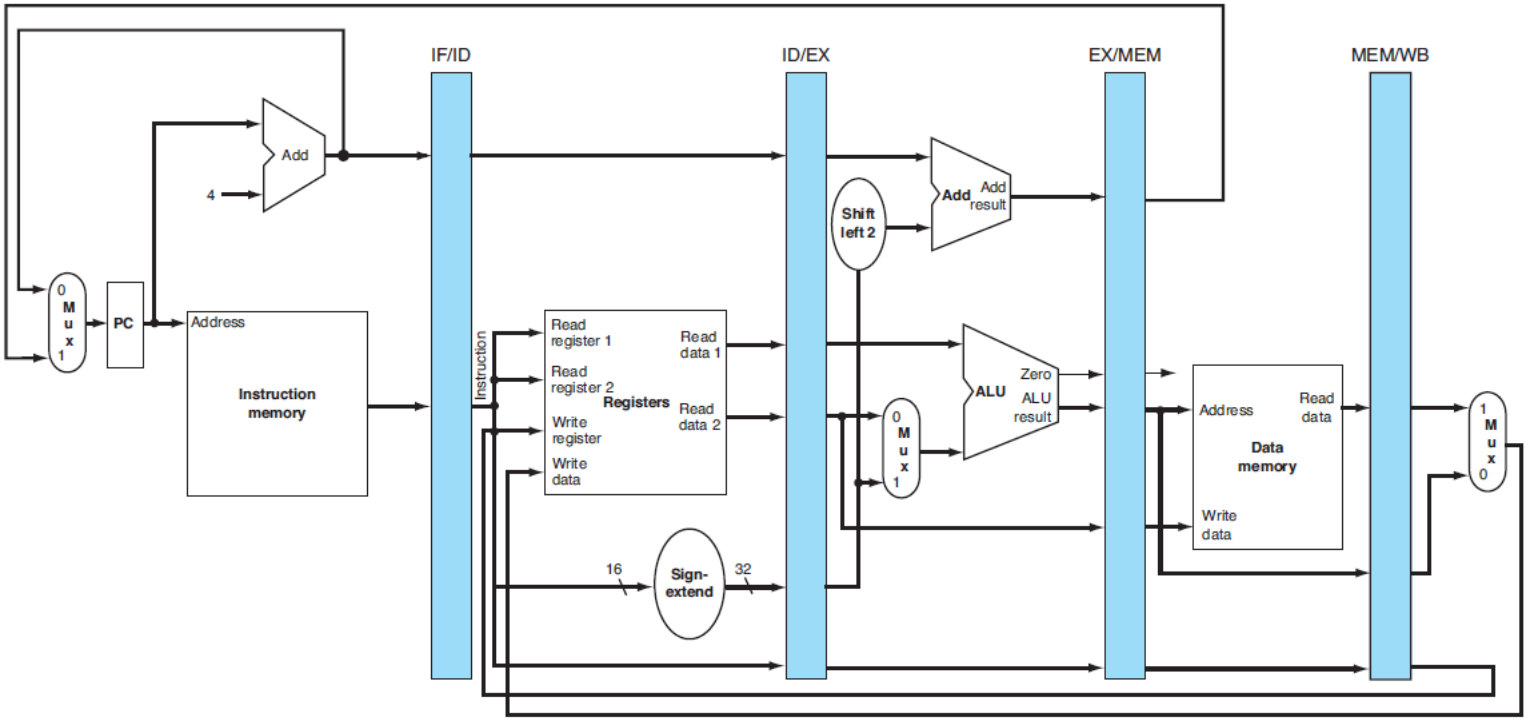
הצגת הרצה בצינור



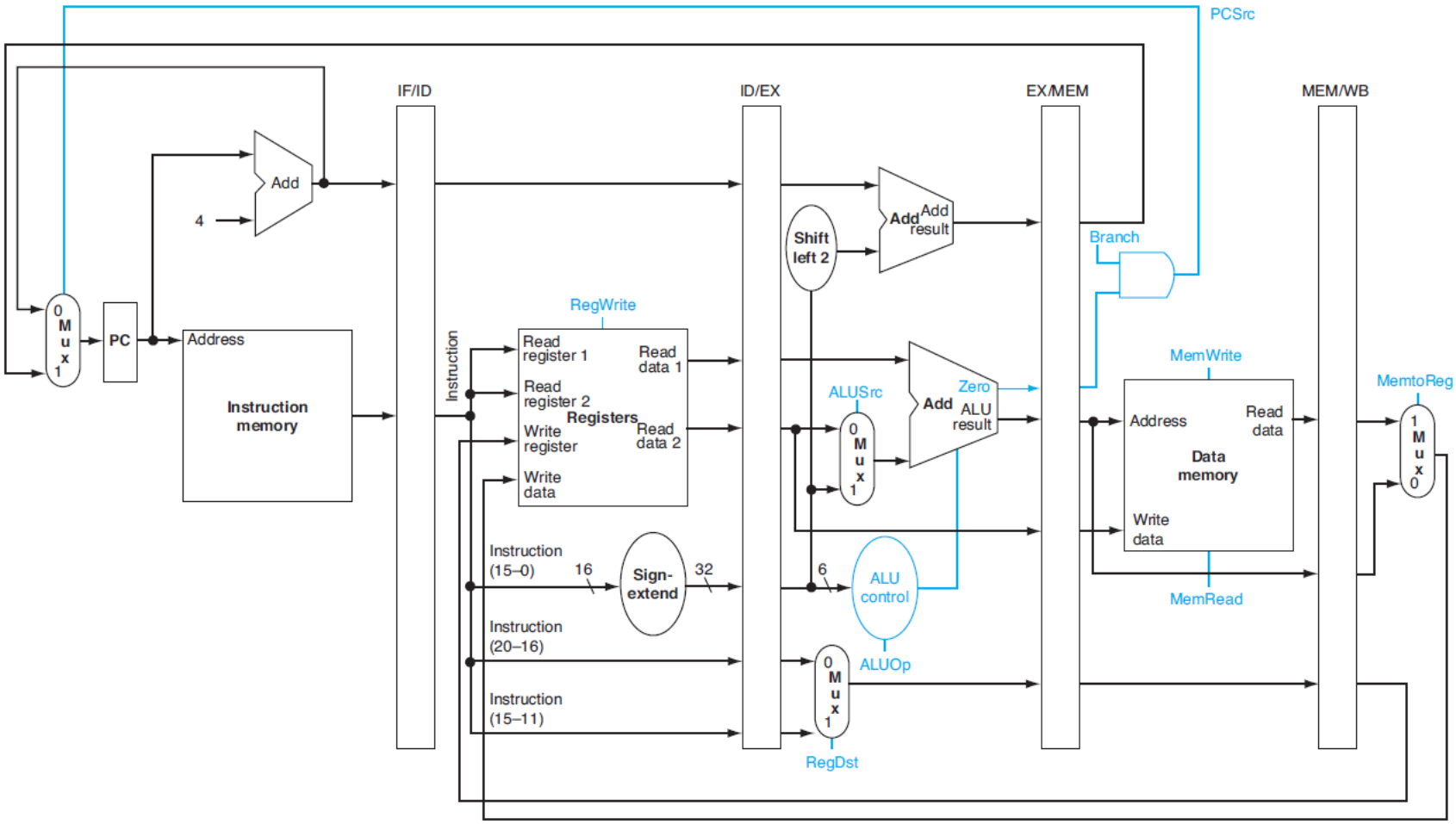
בשרטוט זה ניתן לראות איזה רכיבים עובדים בפועל בכל מחזור שעון לכל פקודה

הצגת הרצה בצינור (במחזור שעון נבחר)

add $14, $5, $6	lw $13, 24 ($1)	add $12, $3, $4	sub $11, $2, $3	lw $10, 20($1)
Instruction fetch	Instruction decode	Execution	Memory	Write-back



הבקר המצונר


PCSrc
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Instruction (15–0)
16
Sign-extend
32
Instruction (20–16)
Instruction (15–11)
Shift left 2
Add
Add result
ALUSrc
Zero
ALU result
6
ALU control
ALUOp
RegDst
Branch
MemWrite
Address
Read data
Data memory
Write data
MemRead
MemtoReg

אותות הבקרה

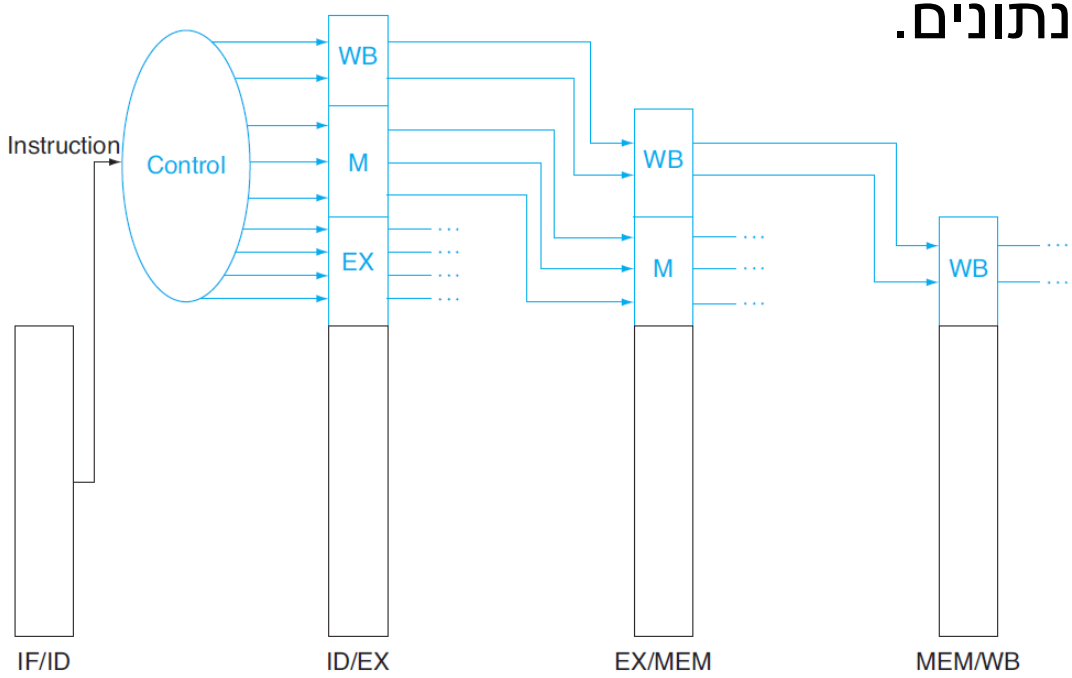
- האותות הנדרשים למימוש מצונר זהים לאותות שהיו במימוש Single-Cycle:

- 3 אותות ששולטים בבוררים השונים
  - RegDst, ALUSrc, MemtoReg
- 3 אותות ששולטים בקריאה וכתיבה
  - RegWrite, MemRead, MemWrite
- אות ששולט במנגנון הקפיצה המותנית
  - נכנס לשער AND עם תוצאת ה-Zero מה-ALU
- אות (2 ביט) שעובר ליחידת הבקרה של ה-ALU

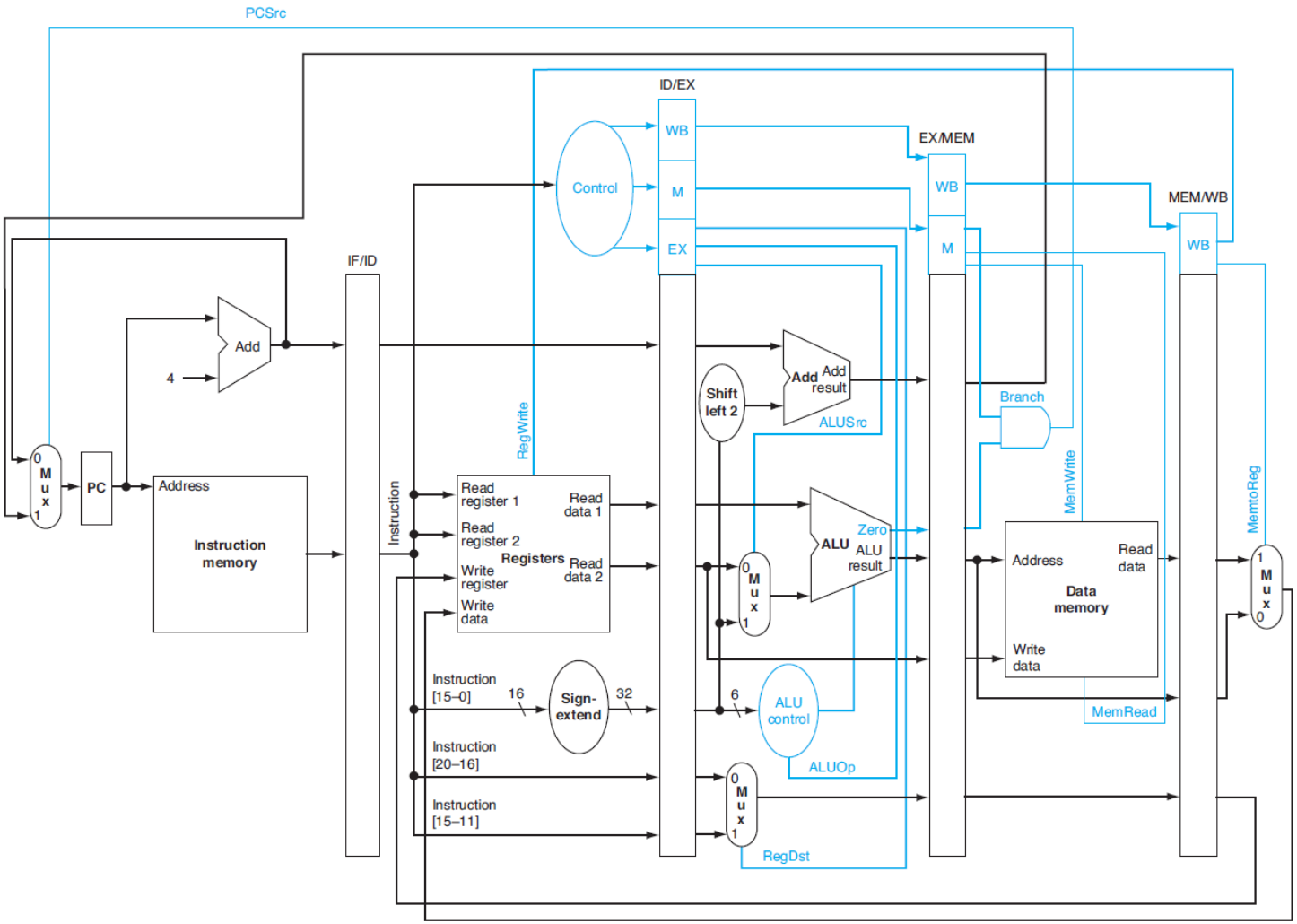
- יחידת הבקרה המשנית ALUcontrol זהה גם היא למימושים הקודמים

הבקר המצונר

- במימוש המצונר כל אות בקרה דרוש רק כאשר ביצוע הפקודה נמצא בשלב הרלוונטי לאות בקרה זה.
- לכן ניתן לצנר את אותות הבקרה, כך שיעברו לשלבים הבאים יחד עם הנתונים.



מסלול הנתונים והבקר המצונר

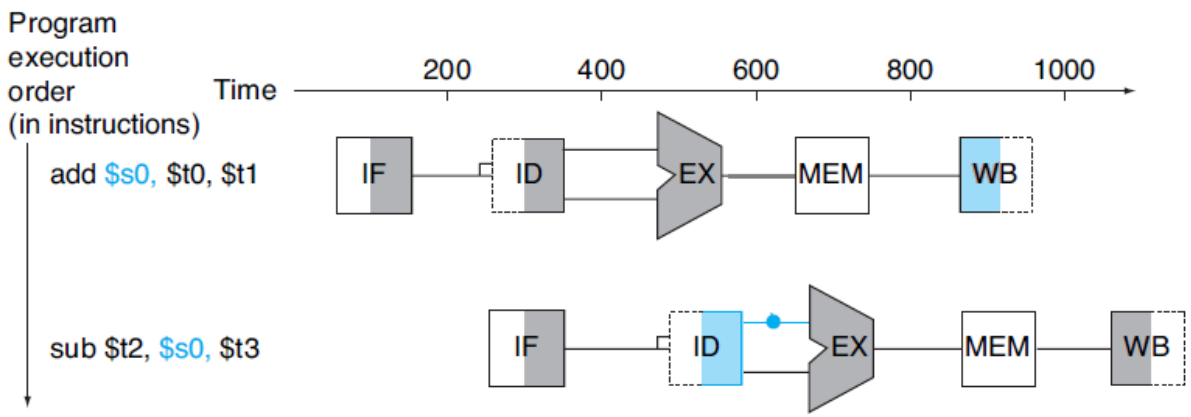


בעיות במימוש מצונר

- הבעיות במימוש מצונר מכונות Hazards ויכולות להיות משני סוגים:
  - Data Hazards: בעיות שנוצרות תלות הביצוע של פקודה בתוצאה של פקודות קודמות שעדיין בשלבי ביצוע בצינור
  - Control Hazards: בעיות המתעוררות כאשר הפקודות אינן מבוצעות לפי סדר איחסונן בזיכרון.
- שני סוגי הבעיות פוגעים ביכולת המעבד להעסיק את כל שלבי מסלול הנתונים כל הזמן. לכן ביצועי המעבד נפגעים.

Data Hazards

- Data Hazards נוצרים כאשר תוצאה של פקודה אחת דרושה לביצוע הפקודה שאחריה בצינור.
  - add $s0, $t0, $t1
  - sub $t2, $s0, $t3
- הפקודה השניה משתמשת בתוצאת החישוב של הפקודה הראשונה לפני שנרשמה לרגיסטר.



פתרונות ל-Data Hazards

- ניתן לפתור את הבעיה על ידי הרחקת הפקודה השניה מהראושנה
  - על ידי הכנסת 3 פקודות שלא עושות כלום (nop)

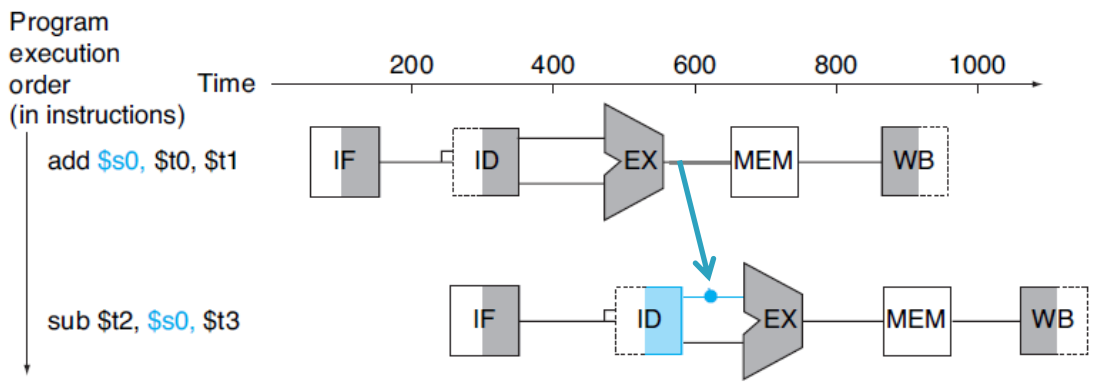
    החסרון: במקרים אלו יש צורך ב-4 מחזורי שעון לביצוע פקודה.

  - על ידי הכנסת 3 פקודות שלא תלויות

- כיוון שרצף פקודות עם תלות אחת בשניה שכיח, כדאי לנסות לפתור בדרך אחרת
  - לבנות מנגנון בחומרה שיאפשר "קיצורי דרך"

פתרונות חומרה ל-Data Hazards

- הפתרון מבוסס על העובדה שכאשר פקודה זקוקה לנתון מפקודה קודמת, יתכן שנתון זה נמצא במקום כלשהו בצינור גם אם לא הסתיים ביצוע הפקודה.
  - בדוגמא שלנו:



- מנגנון זה מכונה Forwarding (עקיפה)

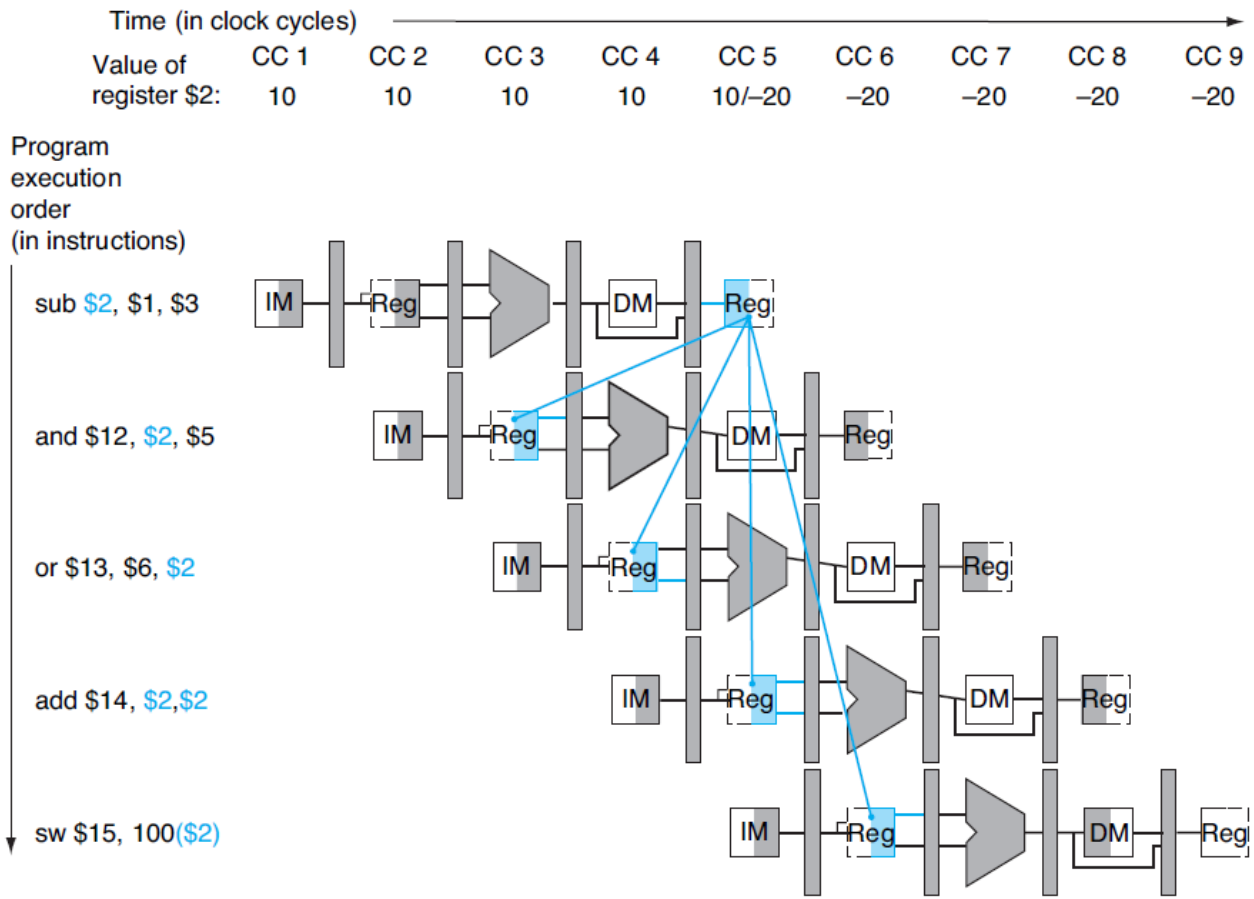
דוגמא נוספת לצורך ב forwarding

- נבחן את רצף הפקודות הבא:

```
sub $2, $1,$3      # Register $2 written by sub
and $12,$2,$5      # 1st operand($2) depends on sub
or $13,$6,$2       # 2nd operand($2) depends on sub
add $14,$2,$2      # 1st($2) & 2nd($2) depend on sub
sw $15,100($2)     # Base ($2) depends on sub
```

- נניח שערך הרגיסטר $2 לפני הפקודה הראשונה היה 10
- לאחר החיסור ערכו 20-.
- המתכנת התכוון שהפקודות הבאות ישתמשו בערך 20-

דוגמא נוספת לצורך ב forwarding



מימוש forwarding

- לצורך מימוש מנגנון העקיפה יש צורך בשני מרכיבים:
  - מנגנון שיזהה את ה-data hazard – יקבע איזו פקודה זקוקה לנתון מאיזו פקודה ויפיק אותות בקרה מתאימים
  - תוספת חומרה למסלול הנתונים (בוררים וחיבורי מעקפים), כך שיהיה אפשרי להביא את הנתונים המתאימים למקומות המתאימים.

זיהוי data hazards

- מנגנון הזיהוי בוחן את הפקודה שנמצאת בשלב EX.
- שתי הפקודות שלפניה (הנמצאות בשלבים MEM ו-WB) עלולות לגרום hazard, במידה והן כותבות לקובץ הרגיסטרים.
- זיהוי כתיבה יעשה על ידי בחינת ערך אות הבקרה RegWrite ברגיסטרים EX/MEM, MEM/WB.
- בנוסף יש לבדוק אם הרגיסטר שעומד להכתב זהה לרגיסטר שמשמש את הפקודה בשלב EX

  - 1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
  - 1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
  - 2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
  - 2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

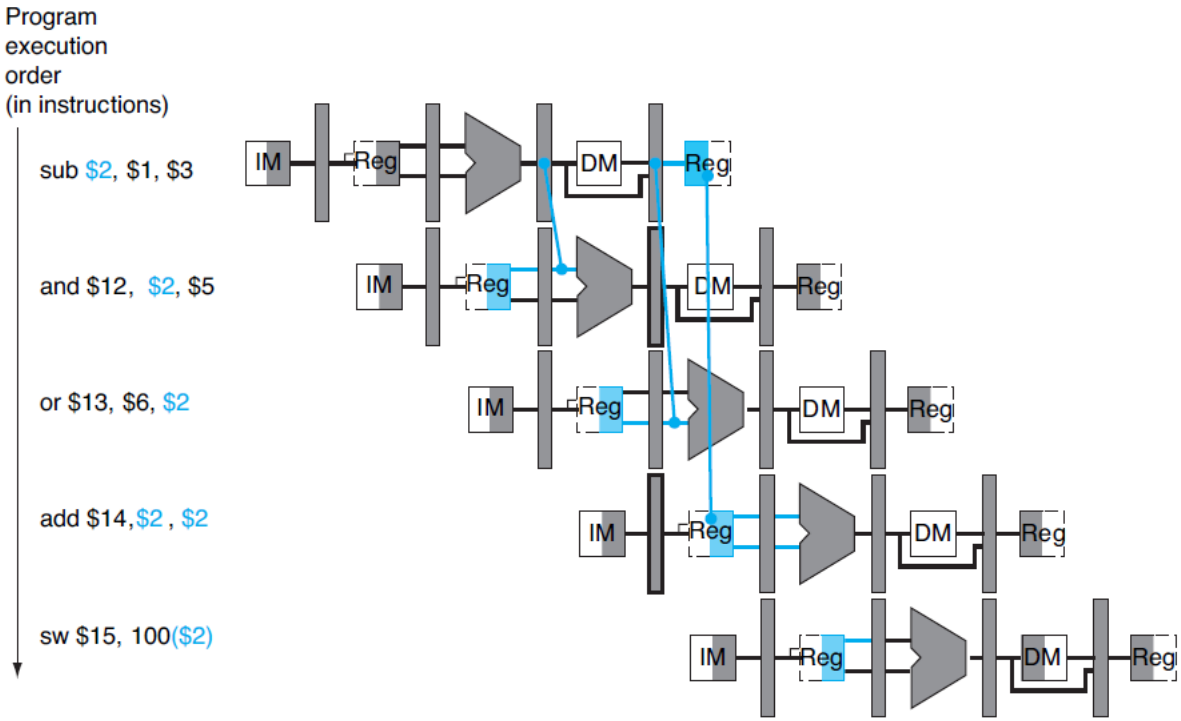
זיהוי data hazards - מקרים מיוחדים

- כאשר הרגיסטר שכותבים\קוראים ממנו הוא רגיסטר 0 אין צורך בעקיפה
- אם כמה מהתנאים מתקיימים:
  - 1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
  - 1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
  - 2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
  - 2b. MEM/WB.RegisterRd = ID/EX.RegisterRt
    - יש לבחור את התנאי שמתאים לפקודה המאוחרת יותר (1)
    - לדוגמא:
  - add $1,$1,$2
  - add $1,$1,$3
  - add $1,$1,$4

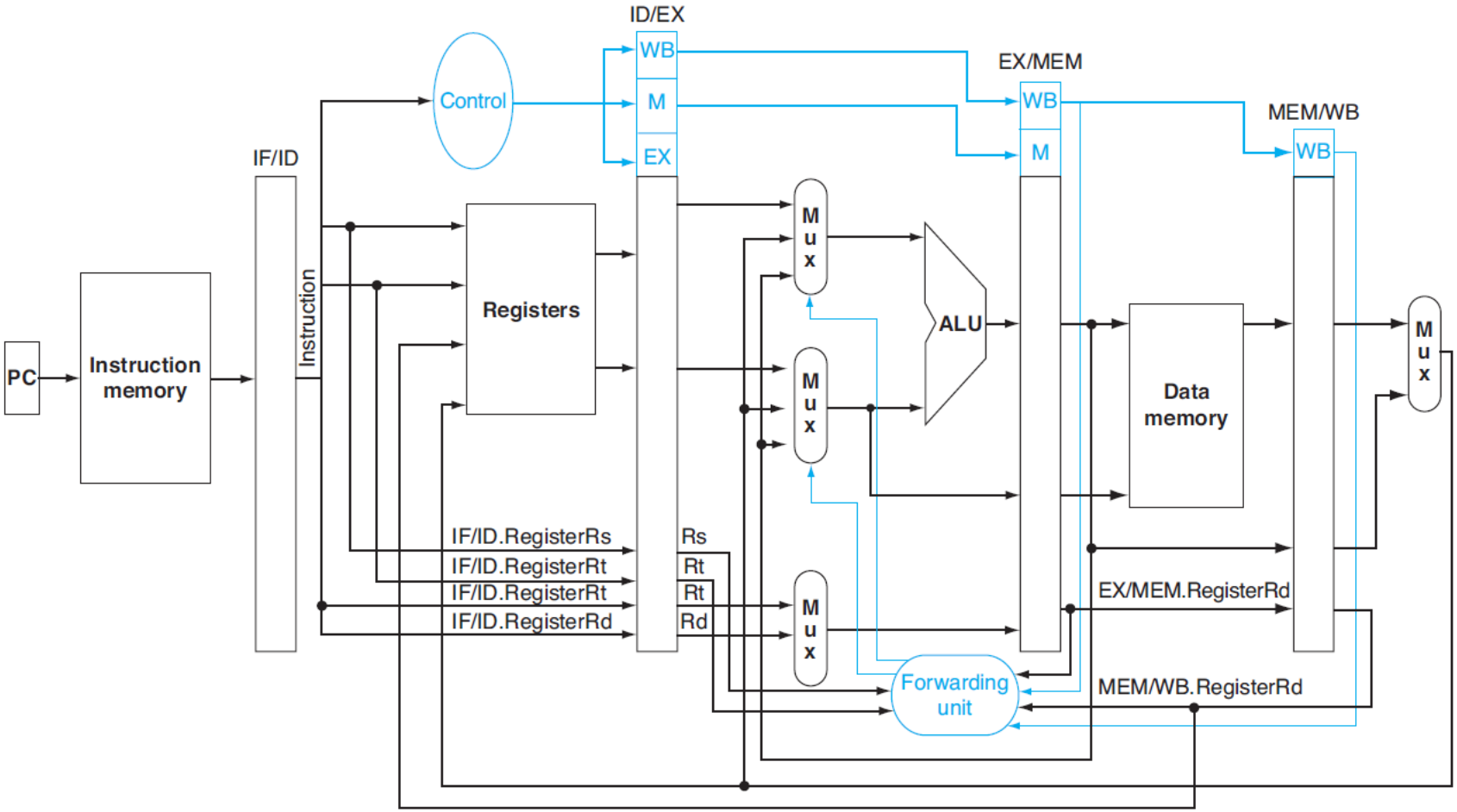
הבאת הנתונים המתאימים

Time (in clock cycles) →

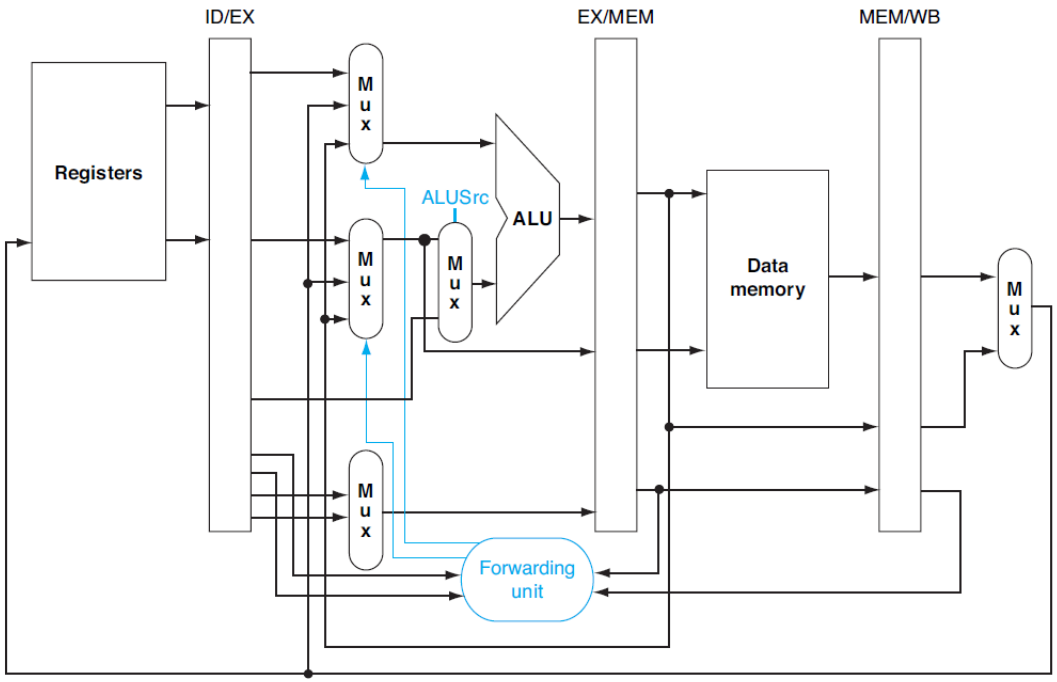
	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2:	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM:	X	X	X	–20	X	X	X	X	X
Value of MEM/WB:	X	X	X	X	–20	X	X	X	X



מימוש forwarding



מימוש forwarding



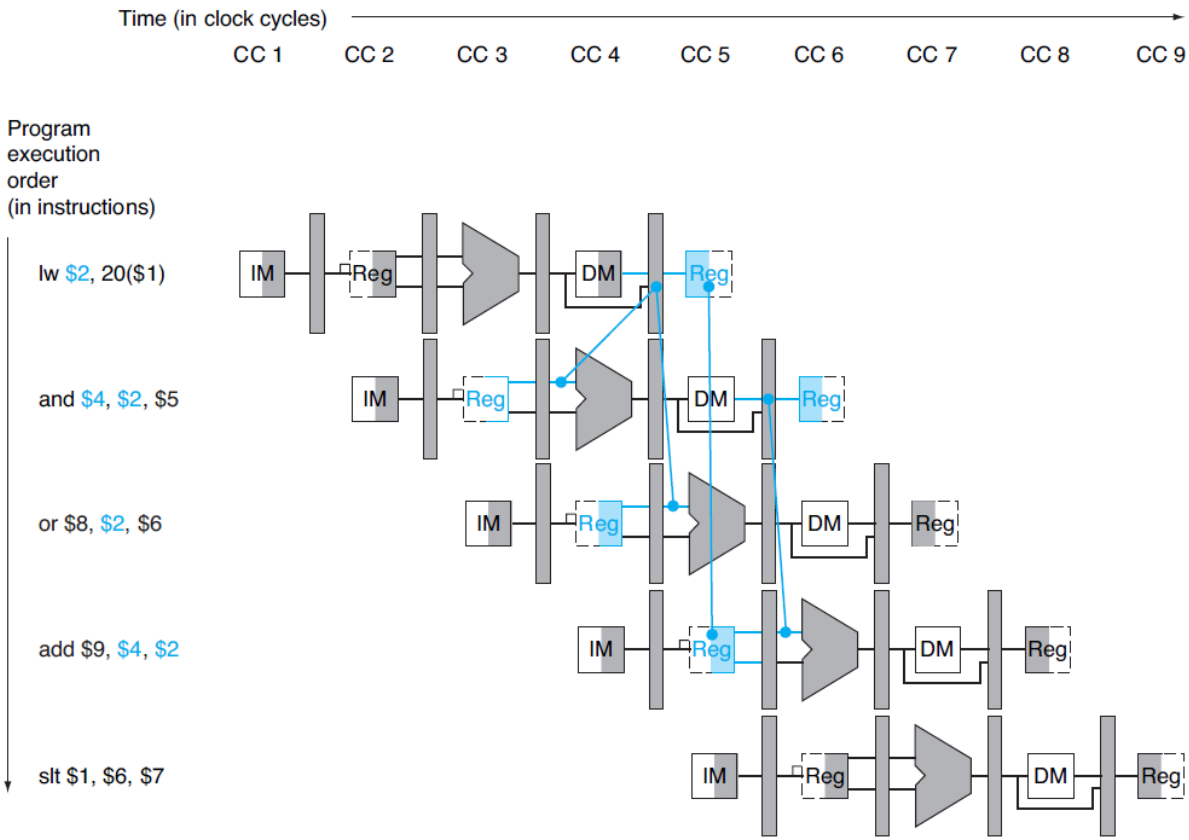
Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

תרגיל

- עבור רצף הפקודות הבא:
  - sub $2, $1,$3
  - and $12,$2,$5
  - or $13,$6,$2
  - add $14,$2,$2
- יש להסביר איזה אותות בקרה יחידת ה forwarding תוציא בכל מחזור שעון בעת ביצוע הפקודות.

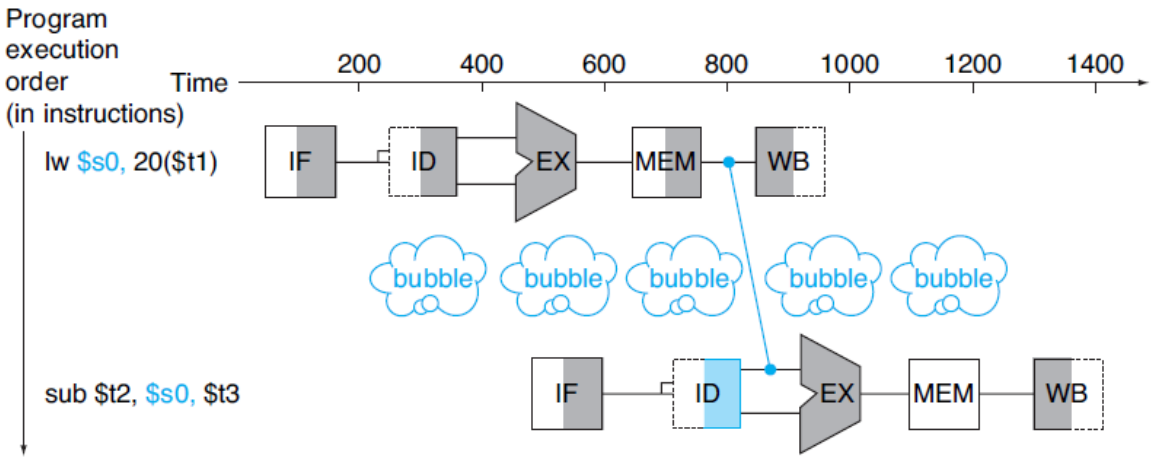
Load hazard

- לא ניתן לבצע forwarding מפקודת lw לפקודה שאחריה כי הנתון הנקרא מהזיכרון זמין רק במחזור MEM



Load hazard

- במקרה זה אין ברירה אלא להכניס עיכוב למחזור שעון אחד



- ניתן לעשות ע"י הקומפיילר שיכניס nop (בועה) במקרה שיש load hazard.
- לאחר עיכוב במחזור אחד ניתן להפעיל את מנגנון העקיפה שיכניס את הנתון ל-ALU

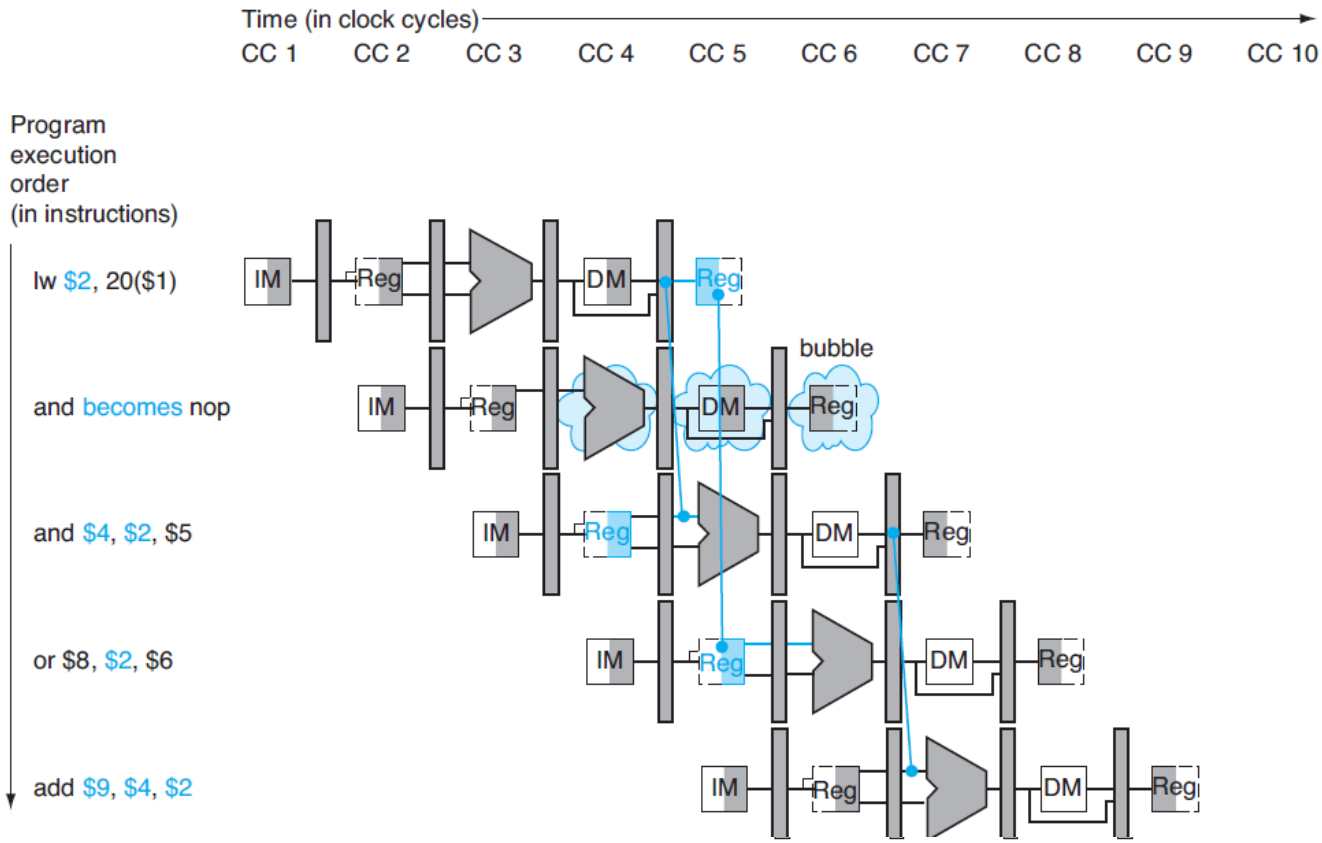
פקודת nop

- ב-MIPS פקודת nop היא פקודת מכונה שכולה אפסים (32 סיביות)
- זוהי פקודה מסוג R שמנסה לכתוב לרגיסטר 0
- רגיסטר 0 הוא "רגיסטר דמה", לכן פקודה זו לא עושה דבר.

טיפול ב-load hazard בחומרה - זיהוי

- במקום שהקומפיילר יכניס nop, נרצה שהחומרה תגלה את הבעיה ותפעל לעכב (stall) את הפקודות הבאות.
- יש צורך במנגנון שבודק אם הפקודה אחרי load קוראת את הרגיסטר אותו ה-load מנסה לכתוב.
- השלב המוקדם ביותר בו ניתן לדעת איזה רגיסטר רוצים לקרוא הוא ID
- אם מתקיים התנאי
  - if (ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
    (ID/EX.RegisterRt = IF/ID.RegisterRt)))
    - מעכבים את הפקודות הבאות במחזור אחד

טיפול ב-load hazard בחומרה - ביצוע

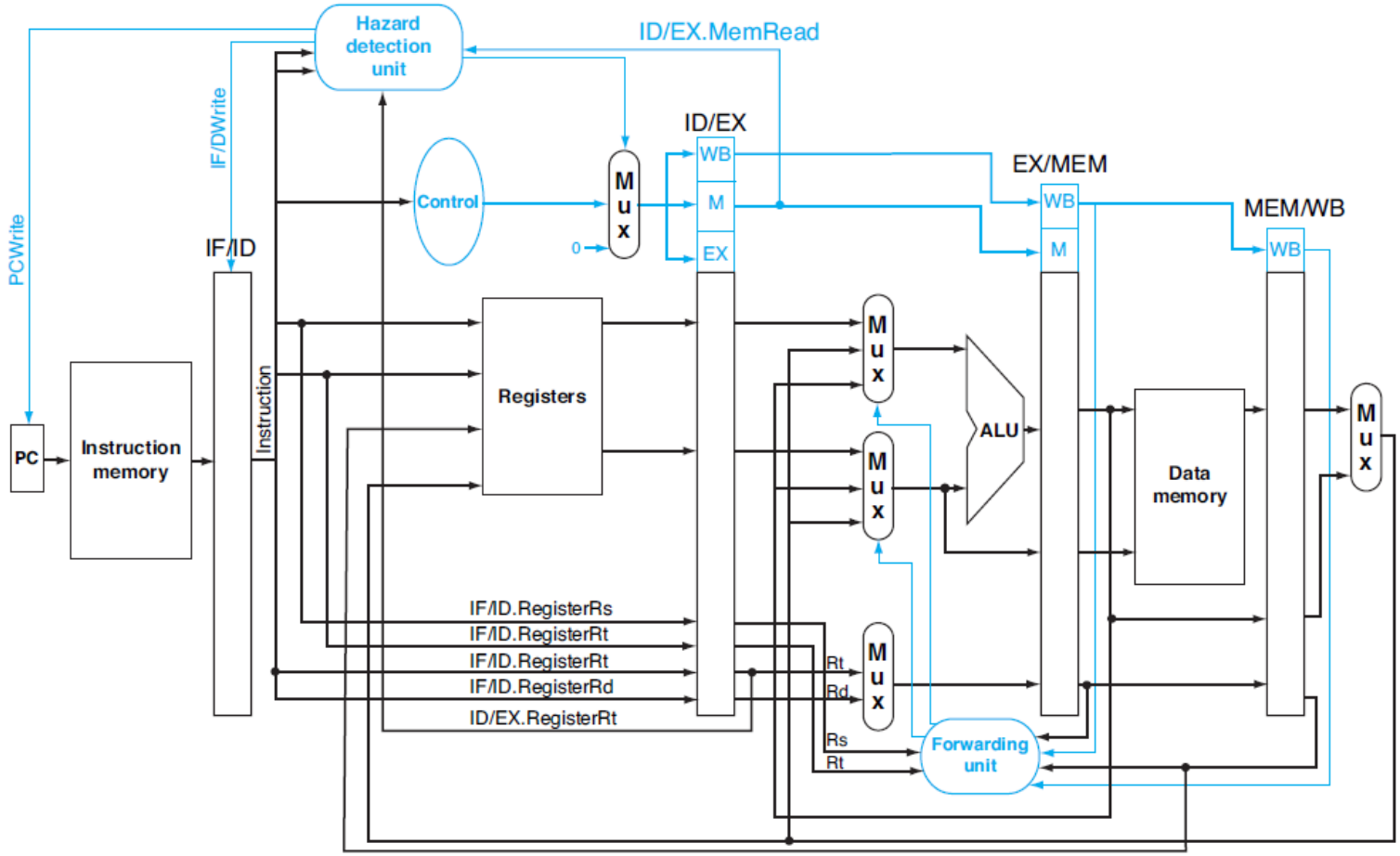


טיפול ב-load hazard בחומרה - ביצוע

- כדי לאפשר עיכוב ב-pipeline יש לבצע מספר פעולות:
  - לא לטעון את הפקודה הבאה
  - לשמור על הפקודה הנוכחית בשלב ID
  - להפוך את הפקודה הנוכחית ל-nop החל משלב EX

- מימוש הפעולות יתבצע בצורה הבאה:
  - הוספת en לרגיסטר PC
  - הוספת en לרגיסטר IF/ID
  - להפוך את אותות הבקרה של הפקודה הנוכחית ל-0 החל משלב EX
    - בצורה זו הרגיסטרים\זיכרון לא ישתנו

הוספת יחידת ה-Hazard detection (Load)



תרגיל

- עבור רצף הפקודות הבא:
  - lw $2, 20($1)
  - and $4,$2,$5
- האם רצף הפקודות יוצר hazard?
- כמה מחזורי שעון נדרשים על מנת להשלים את שתי הפקודות (משלב IF של הפקודה הראשונה עד שלב WB של הפקודה השניה)?
- יש לציין על שרטוט המעגל את האותות הרלוונטים לכל מחזור שעון ולהסביר איך מטופל ה-hazard.

Data hazards נוספים

- ישנם data hazards נוספים שלא דנו בהם,
  - לדוגמא:
    - add $14,$2,$3
    - sw $14, 0($5)
  - כדי לטפל ב-hazard זה יש להוסיף מנגנון שמבצע forwarding לכניסת הזיכרון
- ישנם מקרים נוספים, יש לטפל בכל מקרה לגופו במידת הצורך.

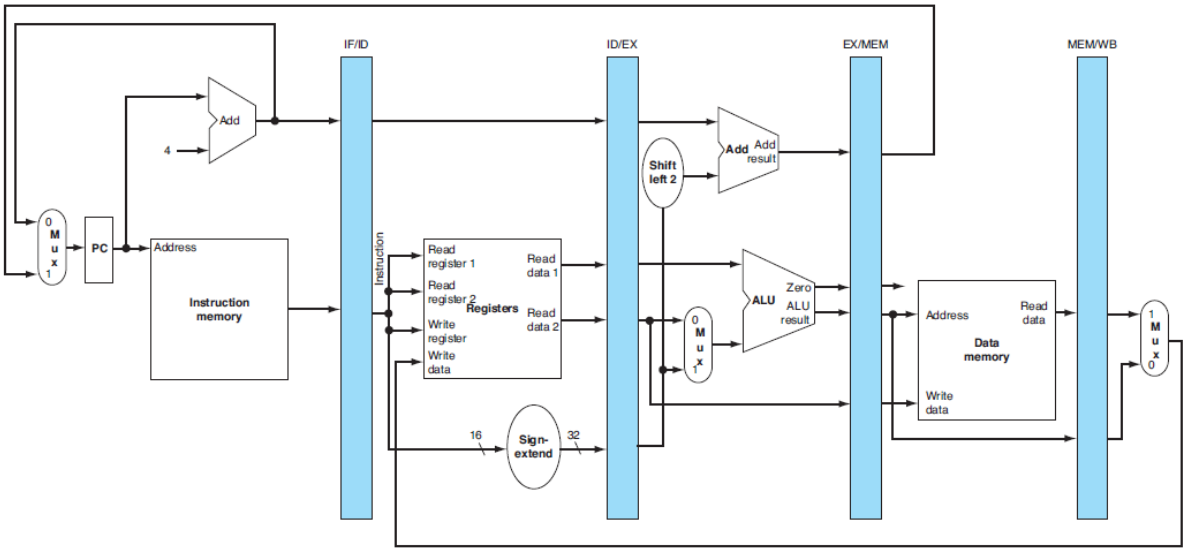
סיכום Data hazards

- data hazards מתרחש כאשר מנסים להשתמש בנתונים לפני שהם זמינים.

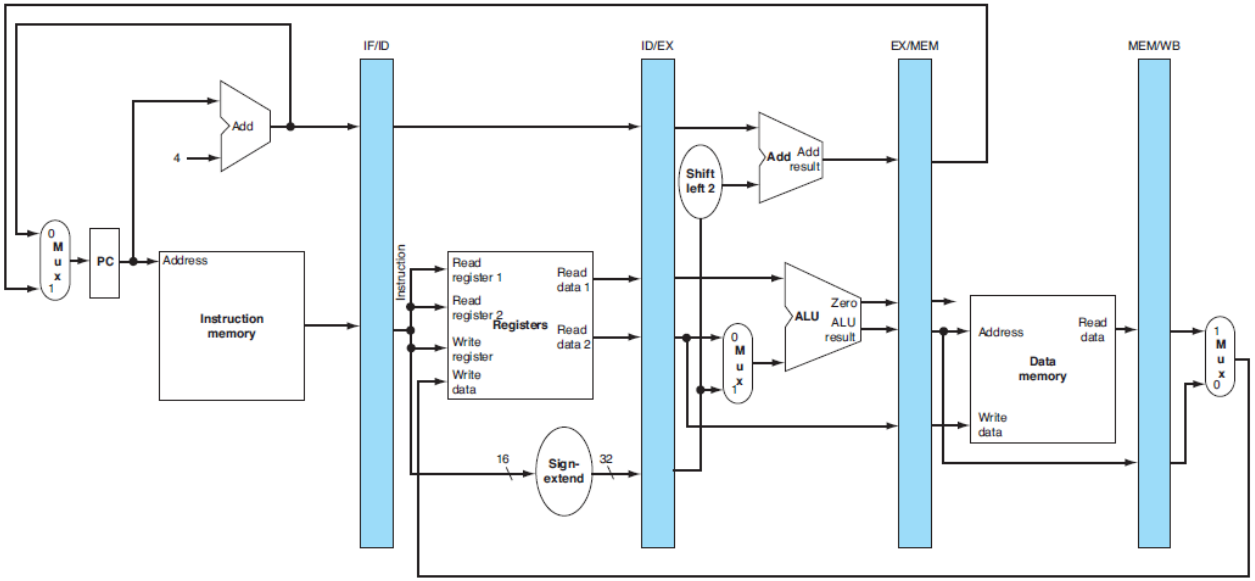
- ניתן לפתור בעיות מסוג זה בכמה דרכים:
  - שינוי סדר הפקודות (Reordering)
  - לעכב את הפקודות הבאות (Stall)
  - שילוב חומרה יעודית
    - מנגנוני עקיפה (Forwarding)
    - זיהוי hazards שלא ניתן לפתור ללא עיכוב (hazard detection)

Control hazards

- Control hazard מתרחש כאשר יש פקודת קפיצה ולא ניתן לטעון פקודות נוספות עד שיודעים את כתובת הפקודה הבאה.
- במימוש המצונר שהוצג, טעינת PC במקרה של קפיצה מתבצעת במחזור השעון הרביעי

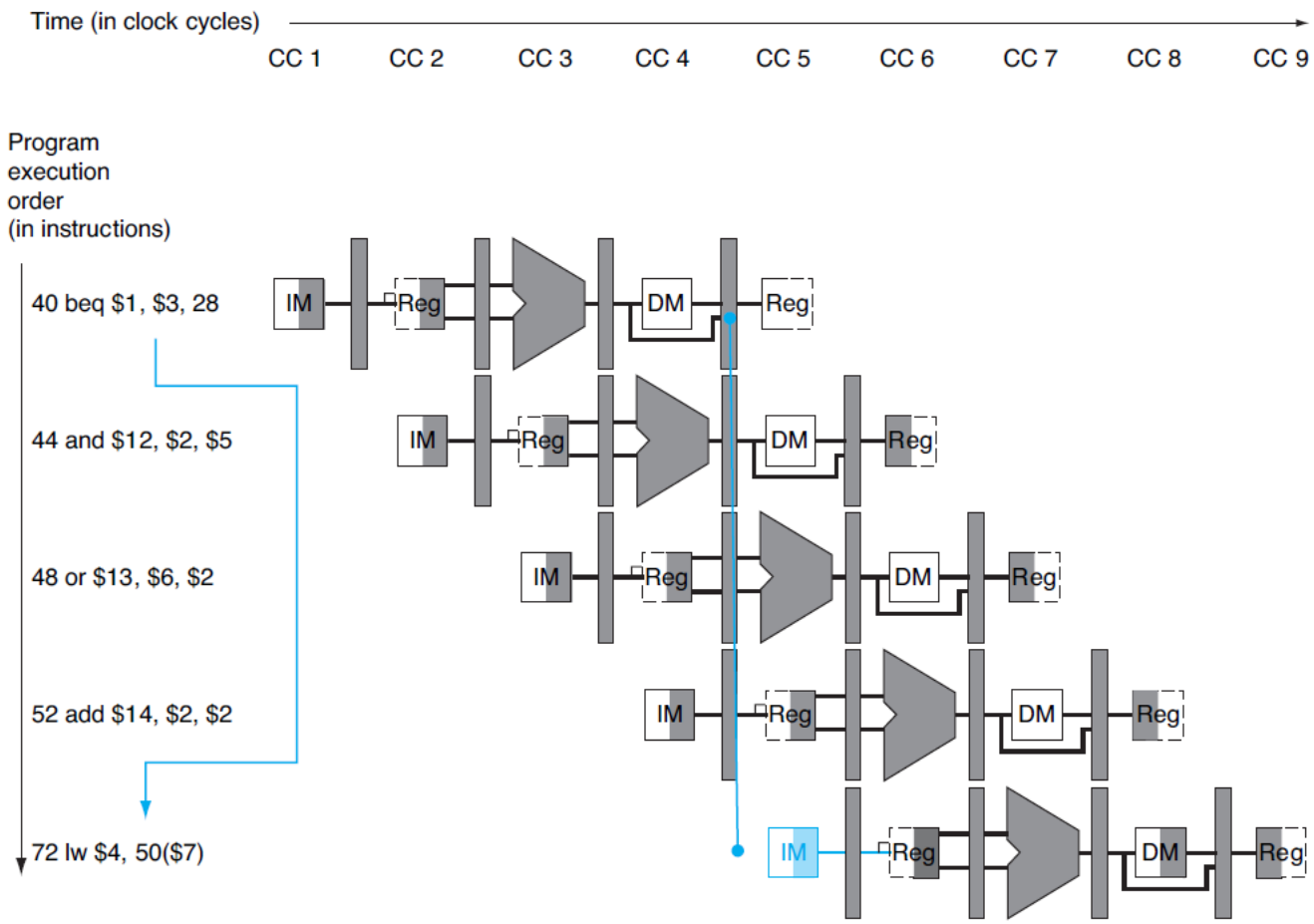


Control hazards



- במקרה של קפיצה מספיקות להכנס לצינור שלוש פקודות שאינן אמורות להתבצע (אם תנאי הקפיצה מתקיים)

דוגמא ל-Control hazard



פתרונות אפשריים ל-Control hazards

- ניתן להכניס 3 פקודות nop לאחר כל קפיצה מותנית
  - יוצר עיכוב מיותר במקרה שלא צריך לקפוץ
- שינוי סדר פקודות
  - אם יש פקודות שבכל מקרה, ללא תלות בקפיצה, צריך לבצע, הקומפיילר יכול להזיזן לשלושת המקומות שאחרי פקודת הקפיצה
- שינוי החומרה והתאמתה לטיפול במקרים אלו
  - במקרה שהקפיצה לא מתבצעת – להמשיך כרגיל
  - אם הקפיצה מתבצעת – לנסות להקטין את מספר ה- branch delay slots.

הנחה שהקפיצה לא מתרחשת

- המנגנון הראשון מסתמך על ההנחה שתנאי הקפיצה לא מתקיים.
  - תחת הנחה זו, ממשיכים לטעון את הפקודות הבאות לפי סדר.
- ההנחה מאפשרת להמשיך כרגיל במקרה שלא צריך לקפוץ בכך חוסך עיכוב מיותר.

- במקרה שההנחה לא מתקיימת יש לרוקן את הצינור מהפקודות שלא צריכות להתבצע.
  - המנגנון דומה לזה שראינו ב-load data hazard, שינוי אותות הבקרה ל-0.
  - כעת יש לדרוס את אותות הבקרה בכל שלבי הצינור שאחרי השלב בו מתקבלת ההחלטה אם צריך לקפוץ (IF, ID, EX).

הקטנת ה-branch delay slots

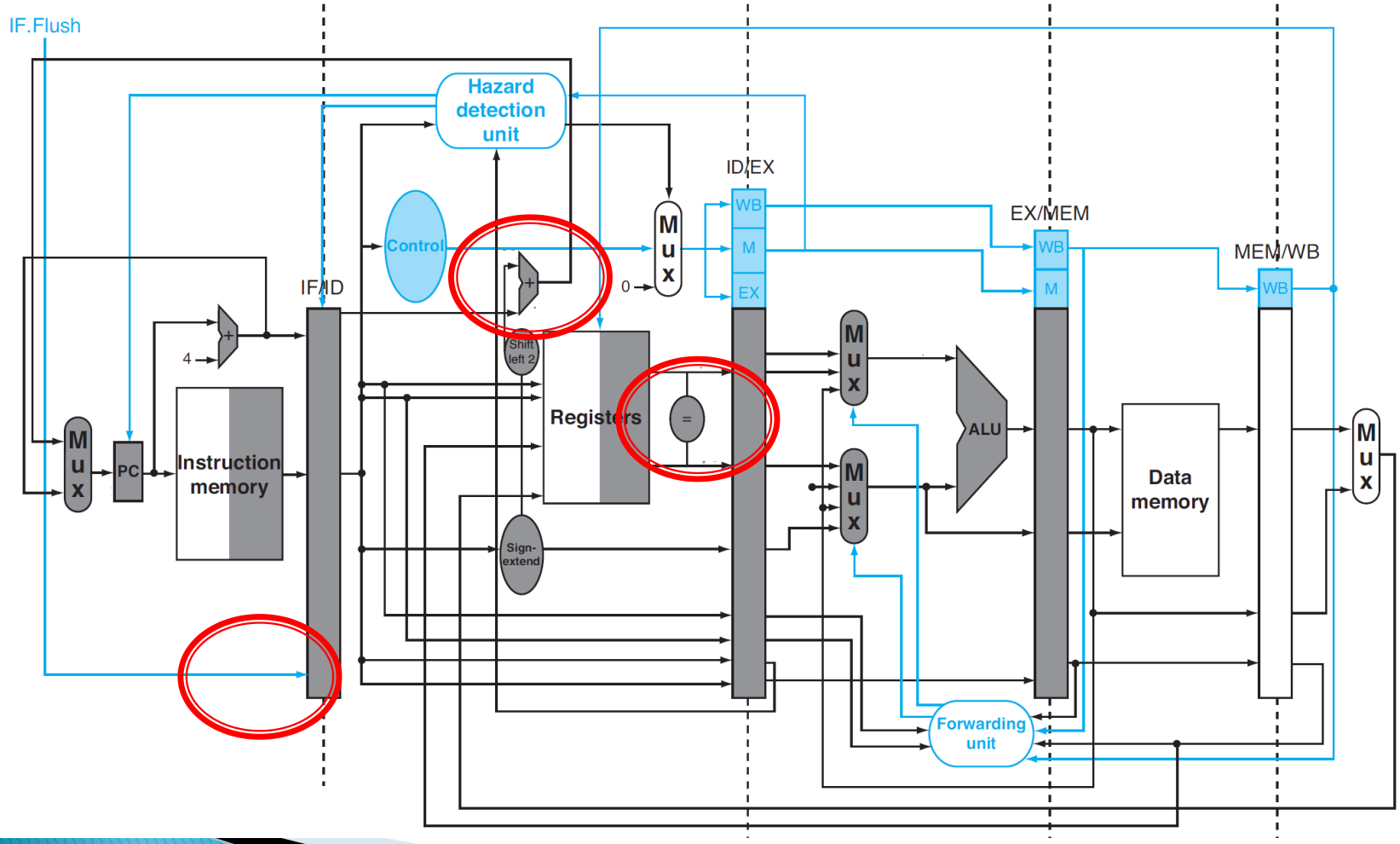
- הרעיון שעומד מאחורי המנגנון:
  - לזהות מוקדם ככל שניתן אם צריך לקפוץ
  - לחשב מוקדם ככל הניתן את כתובת הקפיצה.
- לצורך חישוב כתובת הקפיצה יש צורך ב:
  - כתובת הפקודה הבאה
  - שדה ה-immediate מורחב ומוזז
- שניהם זמינים כבר במחזור השעון השני (ID), ולכן ניתן לחשב את הכתובת כבר בשלב זה.
  - דומה למה שעשינו במימוש ה-multi-cycle

הקטנת ה-branch delay slots

- על מנת לזהות אם צריך לבצע את הקפיצה יש להשוות תוכן שני רגיסטרים.
- תוכן הרגיסטרים ידוע כבר בסוף שלב ה-ID, אך ההשוואה נעשית ב-ALU בשלב EX.

- ניתן להקדים את ההשווה לסוף שלב ID
  - באמצעות שערי XOR ושער OR (אין צורך ב-ALU מלא)
  - ניתן להניח ששלב ID (קריאה מקובץ הרגיסטרים) היה קצר יותר מהשלבים האחרים מלכתחילה, כך שהתוספת לא משנה את זמן המחזור

מעגל לטיפול ב-control hazards



branch delay עם החומרה היעודית

- במקרה שהקפיצה מתבצעת
  - יש להוציא פקודה אחת בלבד מהצינור – יש עיכוב במחזור שעון אחד
  - נעשה על ידי האות IF.Flush
- במקרה שהקפיצה לא מתבצעת
  - ממשיכים כרגיל – ה pipeline לא מושפע.

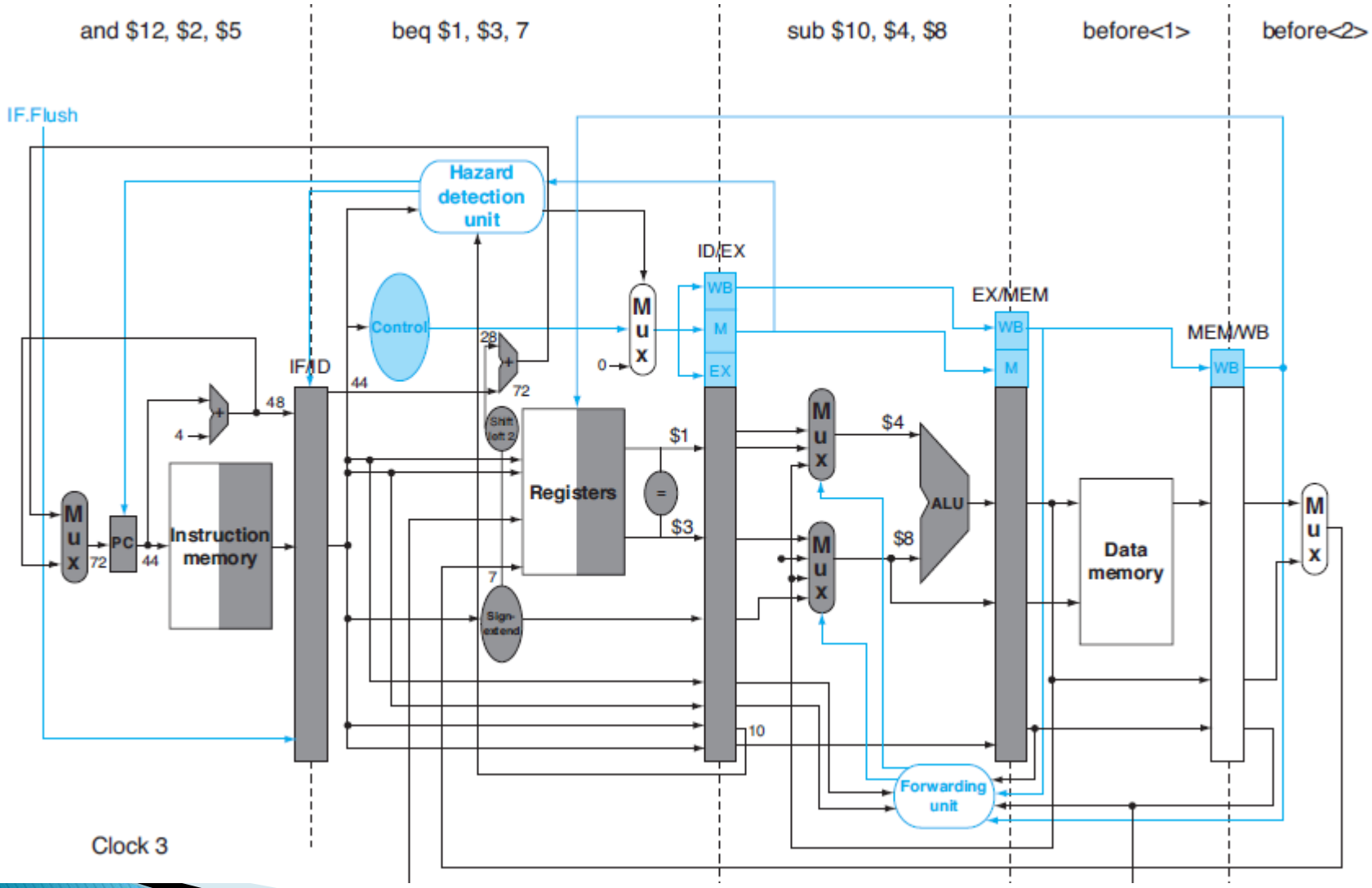
דוגמא

- נתונה התוכנית הבאה

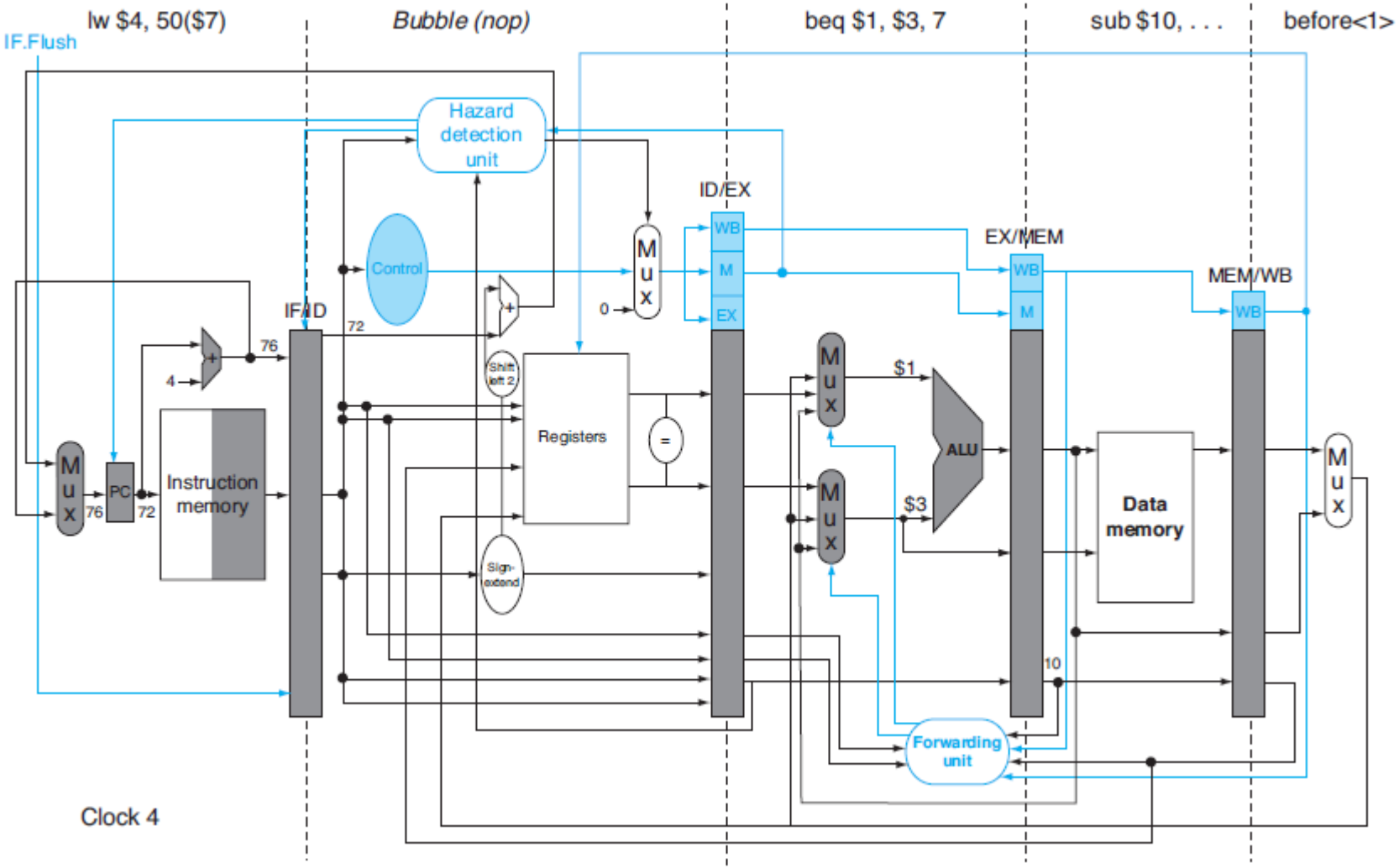
```
36 sub $10, $4, $8
40 beq  $1, $3, 7 # PC-relative branch to 40+4+7*4=72
44 and $12, $2, $5
48 or  $13, $2, $6
52 add $14, $4, $2
56 slt $15, $6, $7
. . .
72 lw  $4,  50($7)
```

- יש להראות מה קורה במעגל כאשר תנאי הקפיצה מתקיים.

דוגמא


and $12, $2, $5
beq $1, $3, 7
sub $10, $4, $8
before<1>
before<2>
IF.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
IF/ID
WB
M
EX
Mux
PC
Instruction memory
Registers
Shift left 2
Sign-extend
ALU
Data memory
Forwarding unit
$1
$3
$4
$8
48
44
72
28
7
10
4
0
Clock 3

דוגמא


lw $4, 50($7)
Bubble (nop)
beq $1, $3, 7
sub $10, . . .
before<1>
IF.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
IF/ID
WB
M
EX
Mux
PC
Instruction memory
Registers
Shift left 2
Sign-extend
ALU
Data memory
Forwarding unit
$1
$3
76
72
4
0
10
Clock 4

בעיה שנוצרת עקב הוספת המנגנון

- מה היה קורה אילו בדוגמא הקודמת התוכנית היתה:

```
36 sub  $1, $4, $8
40 beq  $1, $3, 7 # PC-relative branch to 40+4+7*4=72
44 and $12, $2, $5
48 or  $13, $2, $6
52 add $14, $4, $2
56 slt $15, $6, $7
. . .
72 lw  $4,  50($7)
```

- האם יבוצע בצורה תקינה?
- התשובה שלילית – הערך שמשמש לבדיקת תנאי הקפיצה עדיין לא חושב בזמן ההשוואה
  - במקרים שהנתון מחושב בזמן מאוחר יותר חייבים להכניס מחזורי stall + מנגנון forwarding
  - במקרה של פקודת load שאחריה branch יהיה צורך בשני מחזורי stall

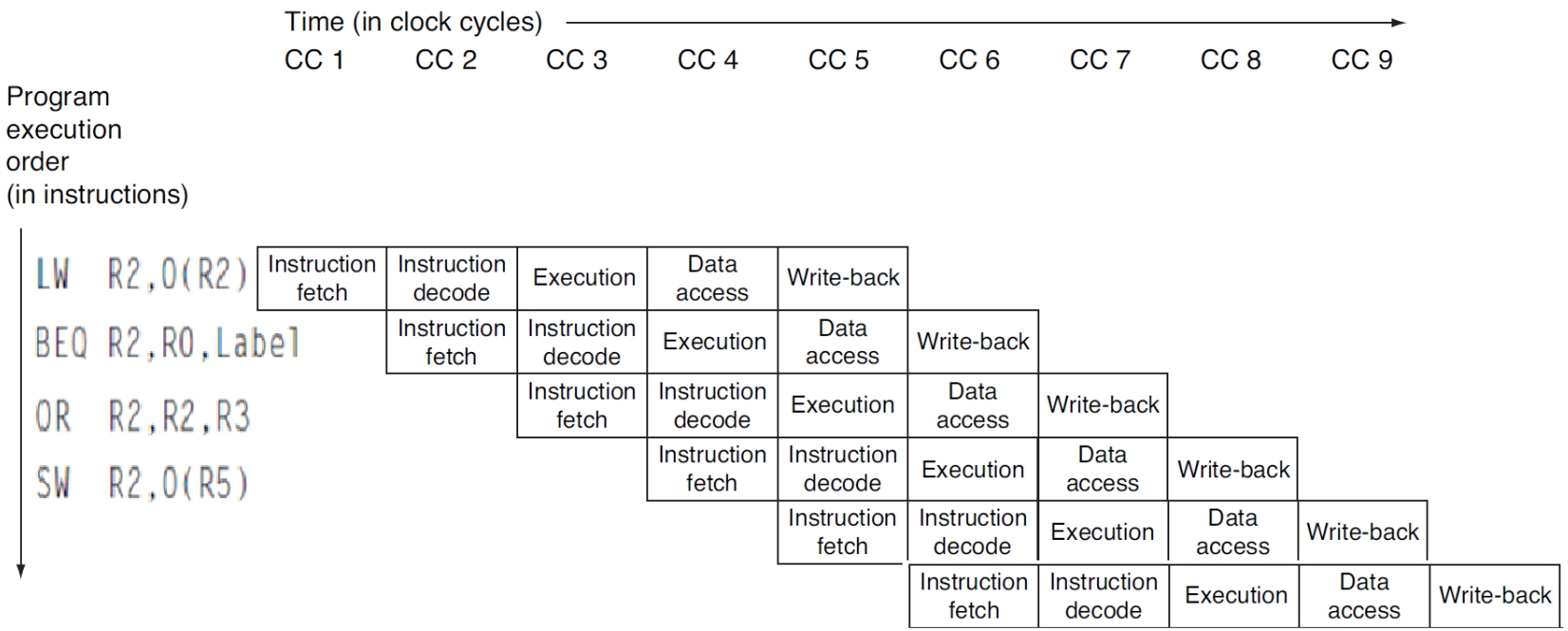
תרגיל

- נתון קטע הקוד הבא:

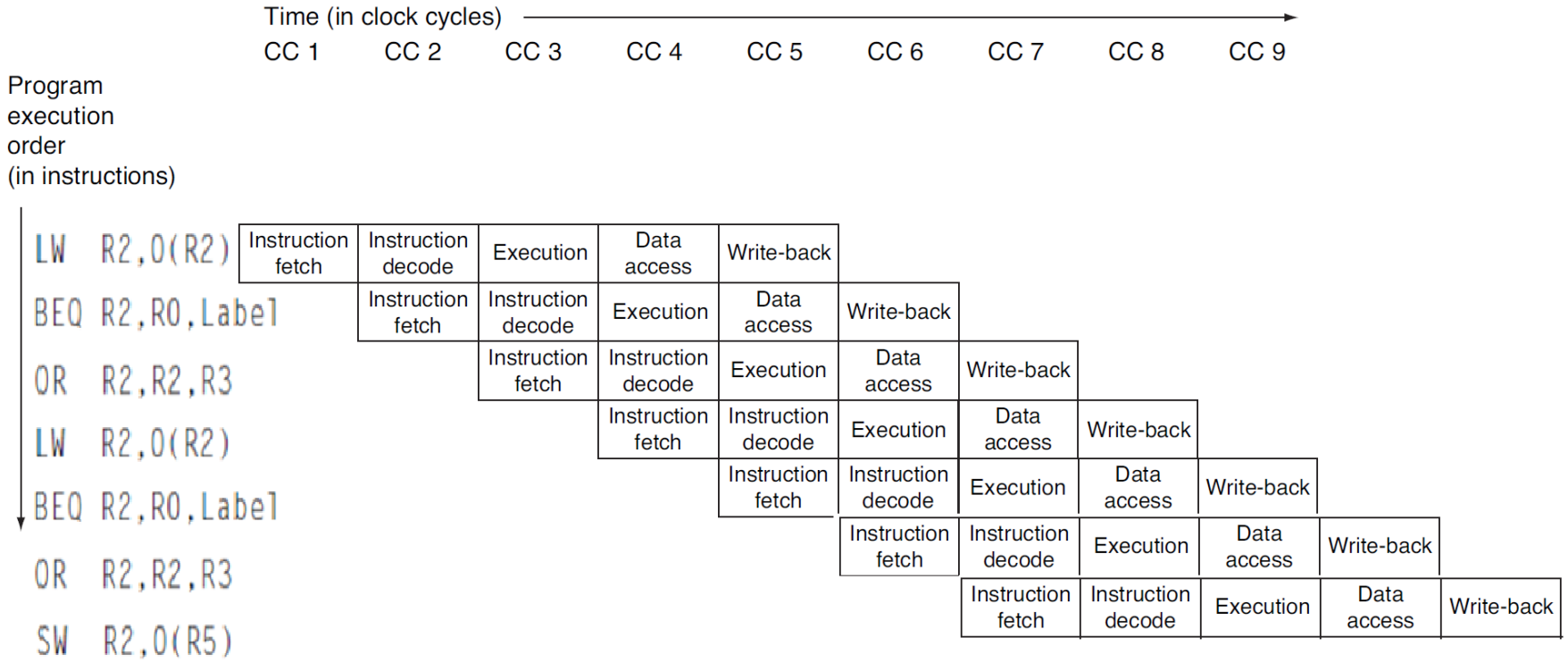
```
Label1: LW   R2,0(R2)
        BEQ  R2,R0,Label ; Taken once, then not taken
        OR   R2,R2,R3
        SW   R2,0(R5)
```

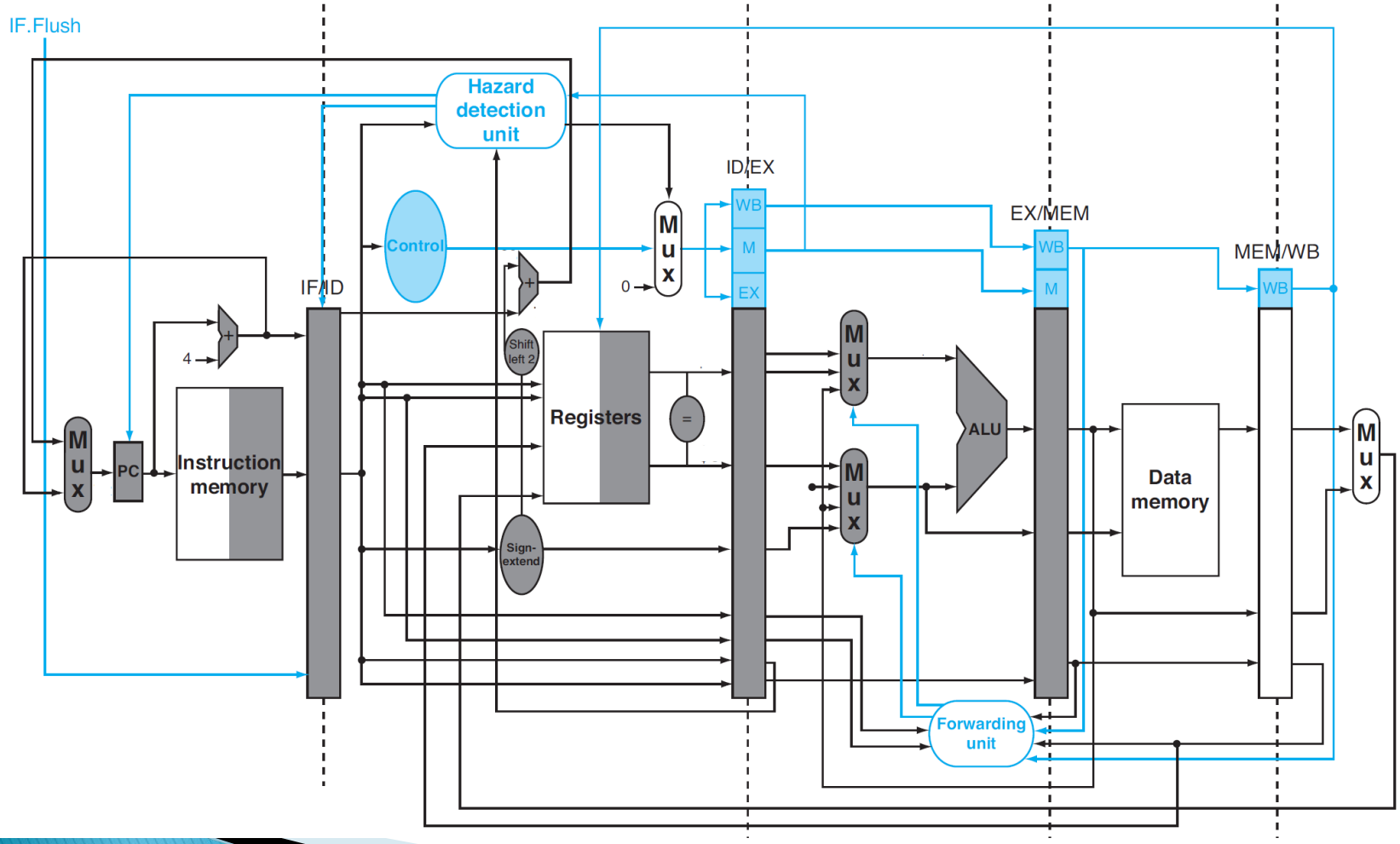
- יש לשרטט דיאגרמת ריצה מצונרת, בהנחה שאין hazards.
- מהו מספר מחזורי stall המינימלי לאחר כל פקודה?
- איזה hazards יש בכל פקודה?
- הסבירו מאיפה ולאן יש לבצע forwarding בכל פקודה.

פתרון


Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Program execution order (in instructions)
LW R2,0(R2)
BEQ R2,R0,Label
OR R2,R2,R3
SW R2,0(R5)
Instruction fetch
Instruction decode
Execution
Data access
Write-back

פתרון


Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Program execution order (in instructions)
LW R2,0(R2)
BEQ R2,R0,Label
OR R2,R2,R3
LW R2,0(R2)
BEQ R2,R0,Label
OR R2,R2,R3
SW R2,0(R5)
Instruction fetch
Instruction decode
Execution
Data access
Write-back


IF.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
IF/ID
WB
M
EX
WB
M
WB
Mux
0
4
PC
Instruction memory
Shift left 2
Registers
=
Sign-extend
ALU
Data memory
Forwarding unit