# תרגול 13

## Pipelining

**אלון שקלר – אוניברסיטת תל אביב**

# למה Pipelining ?

- נרצה שכל רכיב לא יישאר ללא עבודה במהלך מחזור השעון.
- בהנחה שאנו עובדים ב-single cycle לדוגמא, ה-ALU פעיל רק בחלק קטן ממחזור השעון. נרצה לנצלו לטובת פקודות אחרות
- הפתרון: Pipelining - עבודה במקביל על מספר פקודות כאשר רכיב שסיים עבודתו במסגרת פקודה אחת יכול להשתתף בביצוע הפקודה *הבאה*.

# איך Pipelining ?

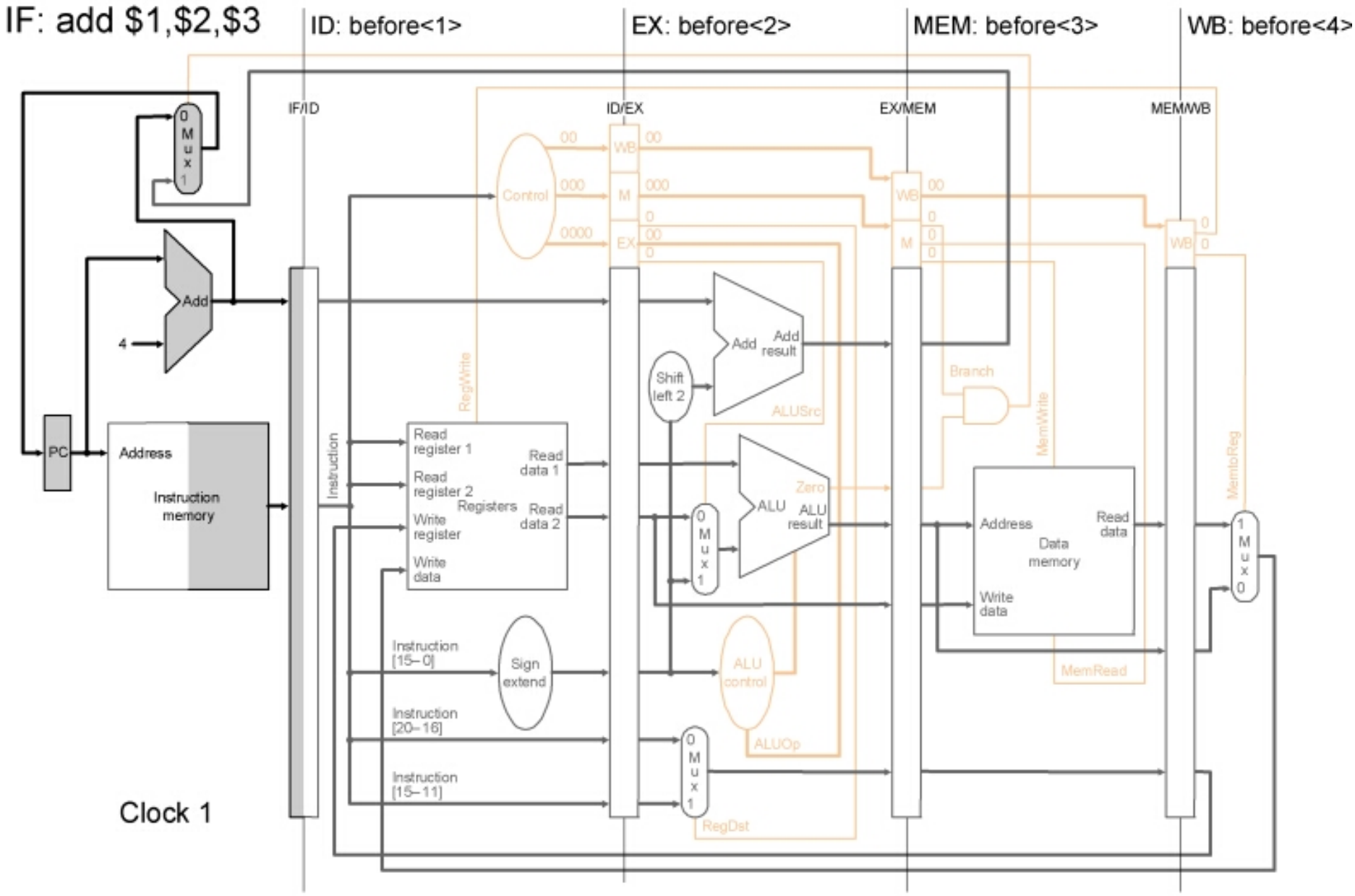
- מחלקים כל פקודה לשלבי עבודה כאשר הפקודה ה"ארוכה" ביותר קובעת את מספר השלבים הכללי (lw לכן 5 שלבים: IF,ID,EX,MEM,WB).
- היות ויש פקודות אשר זקוקות למידע שלהן (כמו רגיסטר יעד) עד סוף ביצוען יש לזכור מידע זה.
- מידע זה נשמר ברגיסטרים מיוחדים אשר שומרים בכל שלב את המידע שנחוץ לשלב הבא.

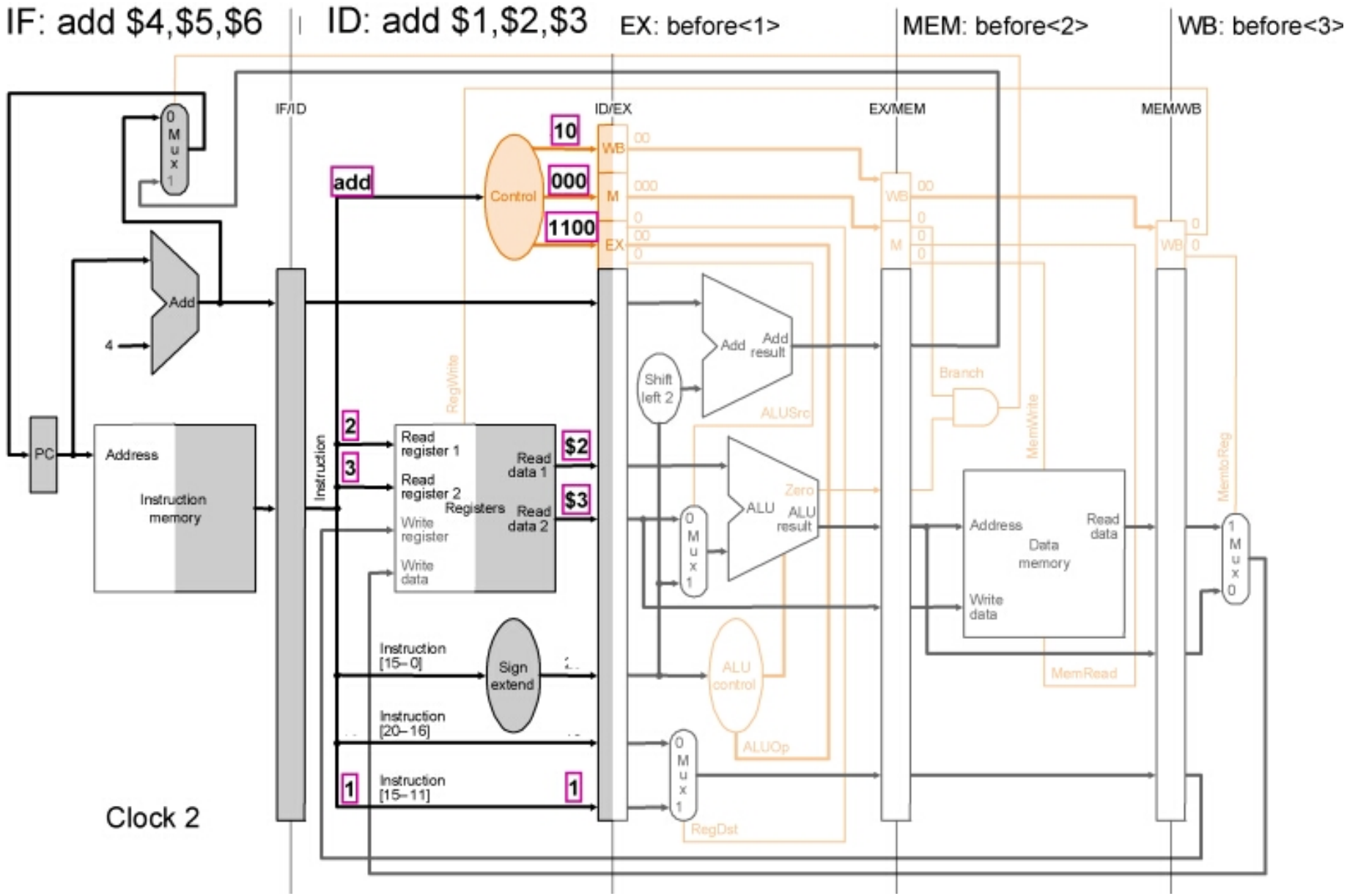
# דוגמא

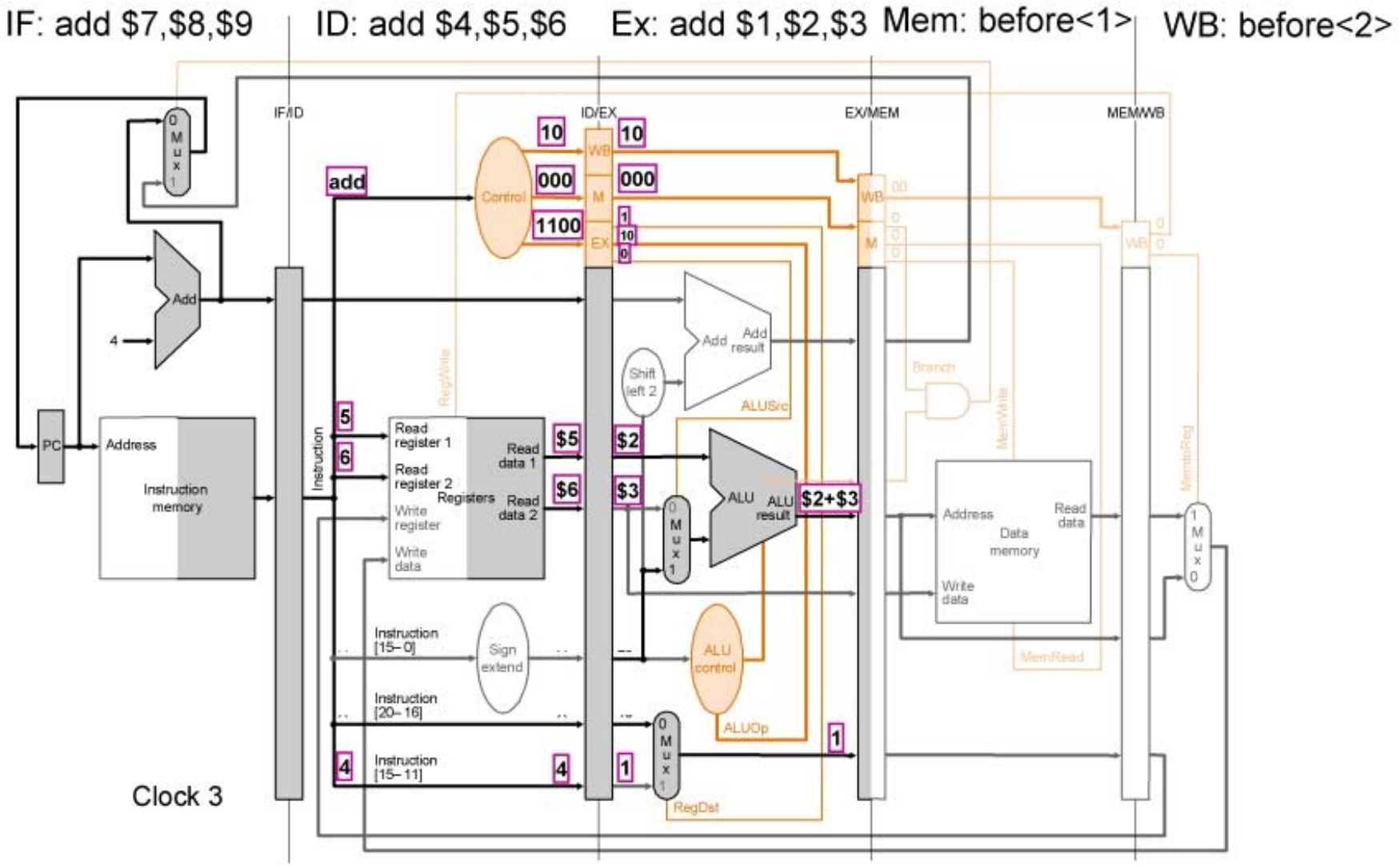
- נתון הקוד הבא:

```
add $1, $2, $3
add $4, $5, $6
add $7, $8, $9
add $10, $11, $12
add $13, $14, $15
```

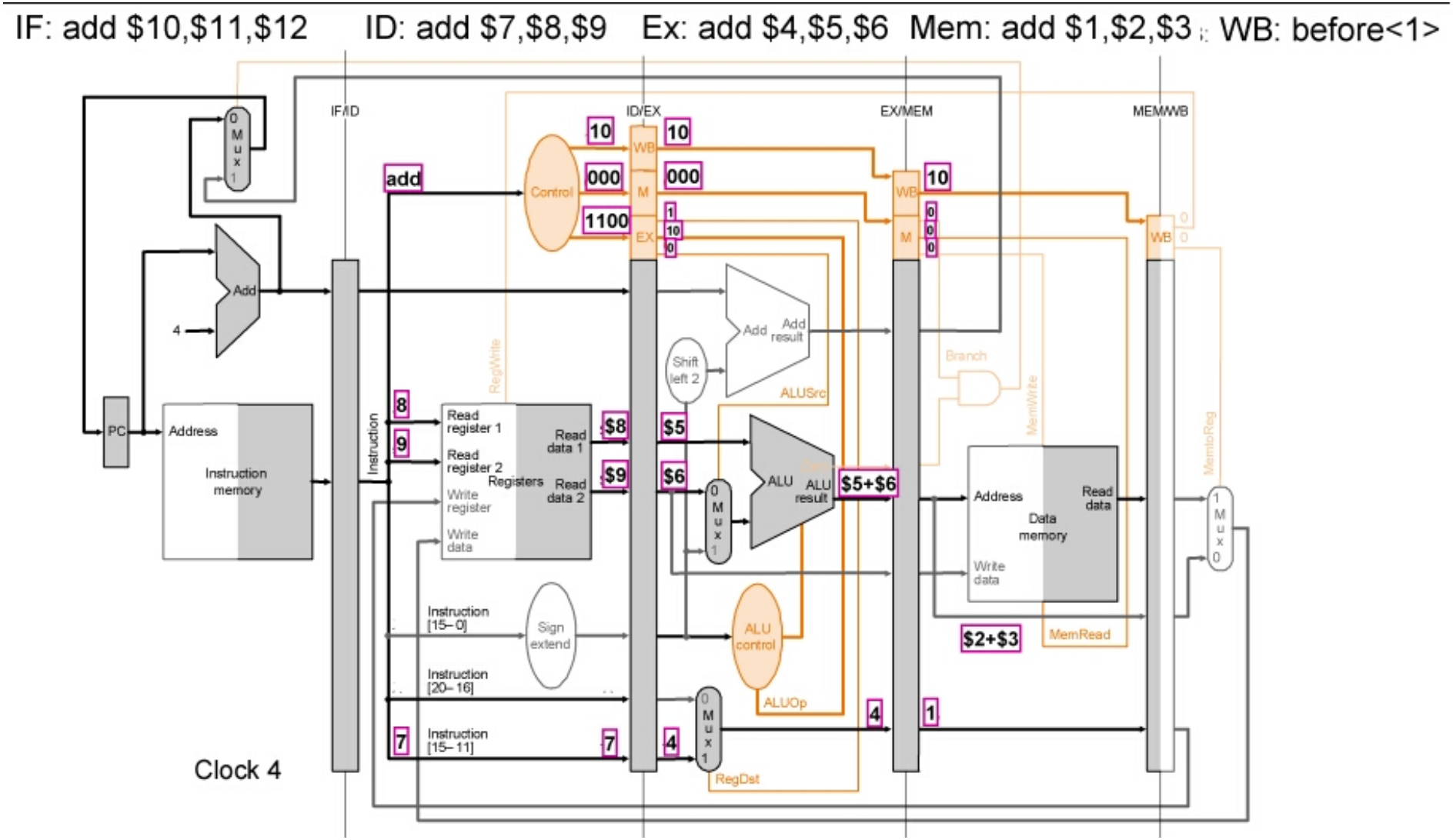
- בסוף ה-cycle החמישי אילו רגיסטרים נכתבים ואילו נקראים ?

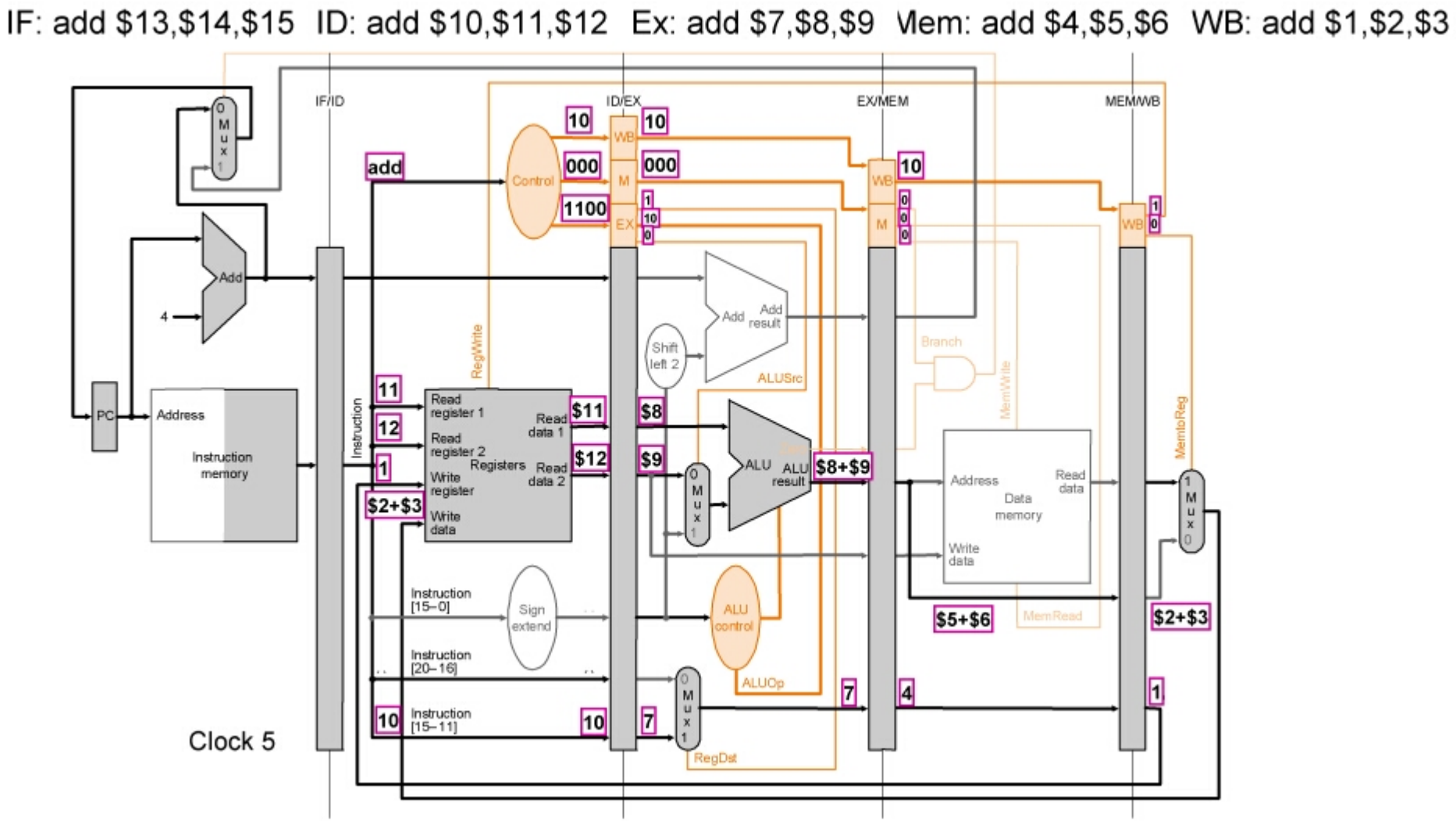







**אלון שקלר – אוניברסיטת תל אביב**






**אלון שקלר – אוניברסיטת תל אביב**

# ערכי הבקרה עבור כל שלב

| Instruction | Execution/Address Calculation stage Control lines | | | | Memory access stage Control lines | | | Write-back stage Control lines | |
|---|---|---|---|---|---|---|---|---|---|
| | Reg Dst | ALU Op1 | ALU Op0 | ALU Src | Branch | Mem Read | Mem Write | Reg Write | Mem to Reg |
| R-format | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| Lw | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| Sw | X | 0 | 0 | 1 | 0 | 0 | 1 | 0 | X |
| Beq | X | 0 | 1 | 0 | 1 | 0 | 0 | 0 | X |

# המשך דוגמא

- בהתייחס לתכנית הקודמת: מה תעשה יחידת ה-forwarding במהלך ה-cycle החמישי. אילו השוואות היא תבצע ?
- פתרון:
  - היחידה בודקת ב- cycle החמישי האם עליה לבצע forwarding ע"י בדיקה הפקודות הרביעית והחמישית אם הן מתכוונות לכתוב למאגר הרגיסטרים והאם הרגיסטר שייכתב משמש כקלט ל-ALU מהפקודה השלישית.
  - ההשוואות שתתבצענה הן:

? 8 = 4? 8 = 1? 9 = 4? 9 = 1

# המשך דוגמא

- בהתייחס לתכנית הקודמת: מה תעשה יחידת ה-hazard detection במהלך ה-cycle החמישי. אילו השוואות היא תבצע ?
- פתרון:
  - היחידה בודקת ב- cycle החמישי האם הפקודה שמתבצעת באותו הרגע ב-ALU היא lw והאם הפקודה שבשלב ה-ID קוראת מהרגיסטר שאליו פקודת ה-lw כותבת.
  - אם כן יש צורך ב-stall.
  - אם הייתה הפקודה השלישית lw אז היחידה הייתה בודקת אם רגיסטר היעד שלה הוא 11 או 12.

# דוגמא נוספת

- נתונות הפקודות הבאות

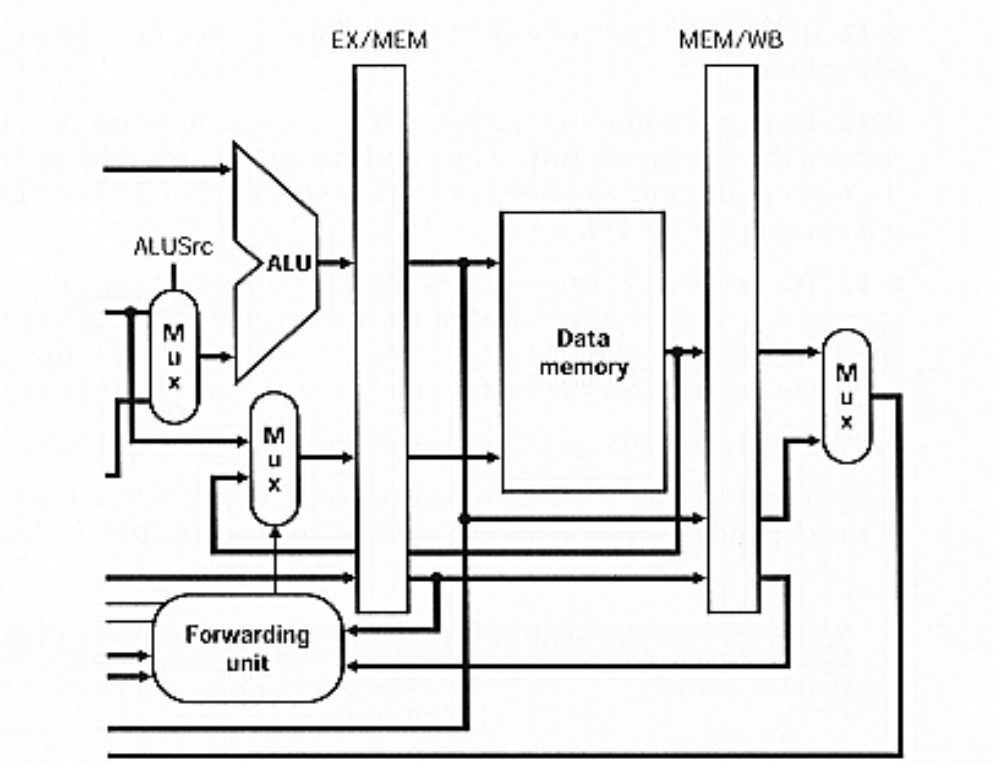
lw $2, 100($5)

sw $2, 200($6)

- אילו שינויים בחומרה datapath יש להכניס על-מנת ששתי פקודות אלו תעבודנה ללא stall ?
- שני פתרונות:
  - פתרון א - בודק את השילוב lw-sw כאשר פקודת ה-lw נמצאת בשלב ה-MEM ופקודת ה-sw נמצאת בשלב ה-EX.
  - פתרון ב - בודק את השילוב lw-sw כאשר פקודת ה-lw נמצאת בשלב ה-WB ופקודת ה-sw נמצאת בשלב ה-MEM.

Solution is courtesy of Alex Actipis -Textbook Sales Representative - ELSEVIER SCIENCE

# פתרון - א

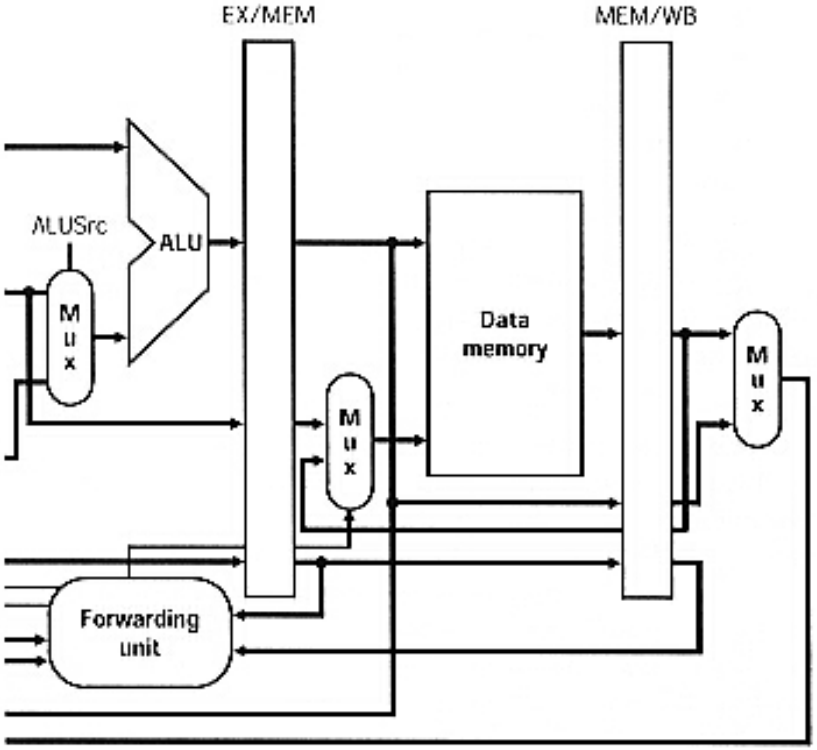



```
if    (ID/EX.MemWrite and                 // sw in EX stage?
       EX/MEM.MemRead and                 // lw in MEM stage?
       (ID/EX.RegisterRt = EX/MEM.RegisterRd) and
                                          // same register?
       (EX/MEM.RegisterRd ≠ 0))           // but not r0?
then
       Mux = 1                            // forward lw value
else
       Mux = 0                            // do not forward
```

Actual Instruction and not a FLUSH

# פתרון - ב

Solution is courtesy of Alex Actipis -Textbook Sales Representative - ELSEVIER SCIENCE




```
if (EX/MEM.MemWrite and                    // sw in MEM stage?
   (MEM/WB.MemToReg = 1) and MEM/WB.RegWrite and
                                           // lw in WB stage?
   (EX/MEM.RegisterRd = MEM/WB.RegisterRd) and
                                           // same register?
   (MEM/WB.RegisterRd ≠ 0))                // but not r0?
then
   Mux = 1                                 // forward lw value
else
   Mux = 0                                 // do not forward
```

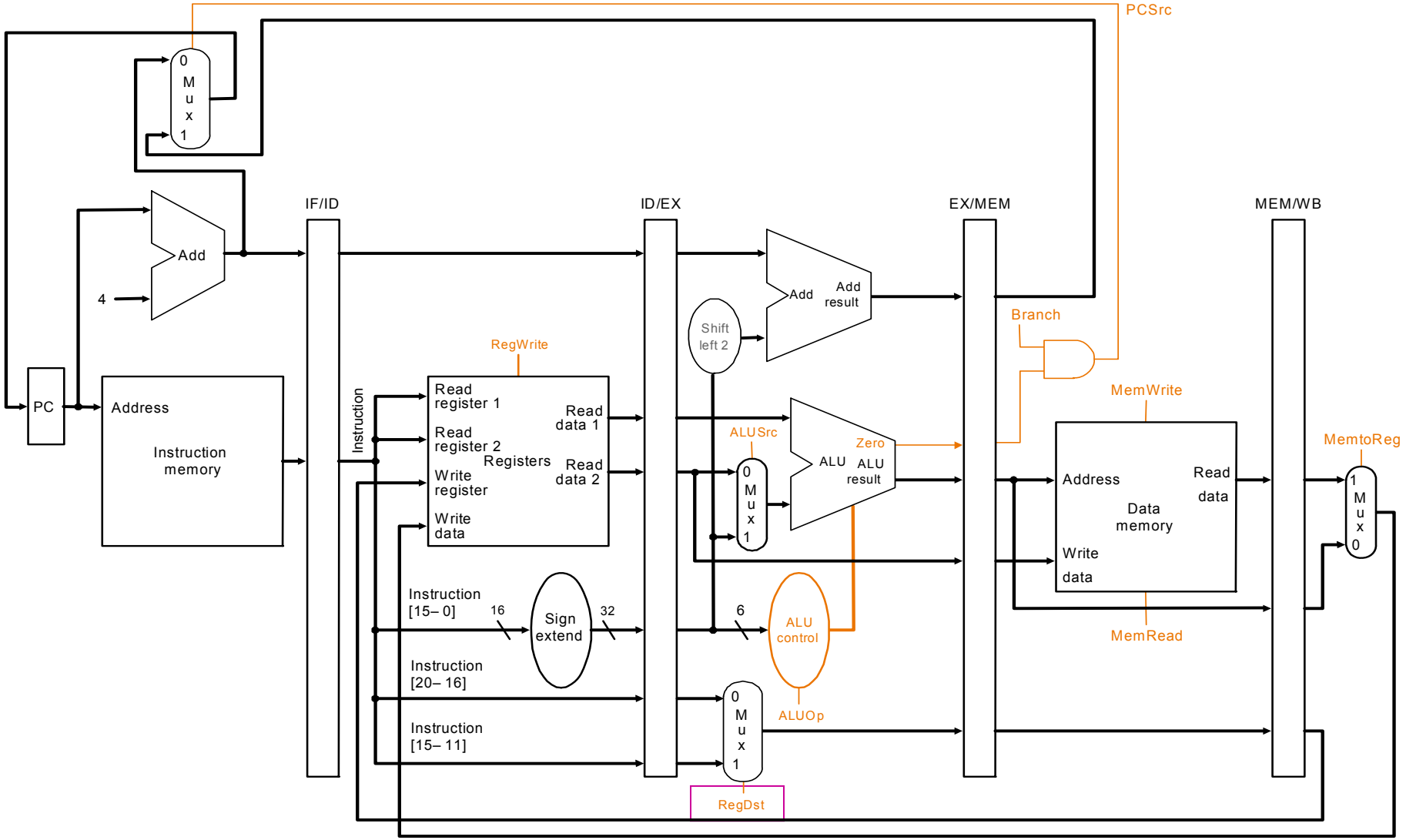
LW is identified differently because the loss of the MEM control signals

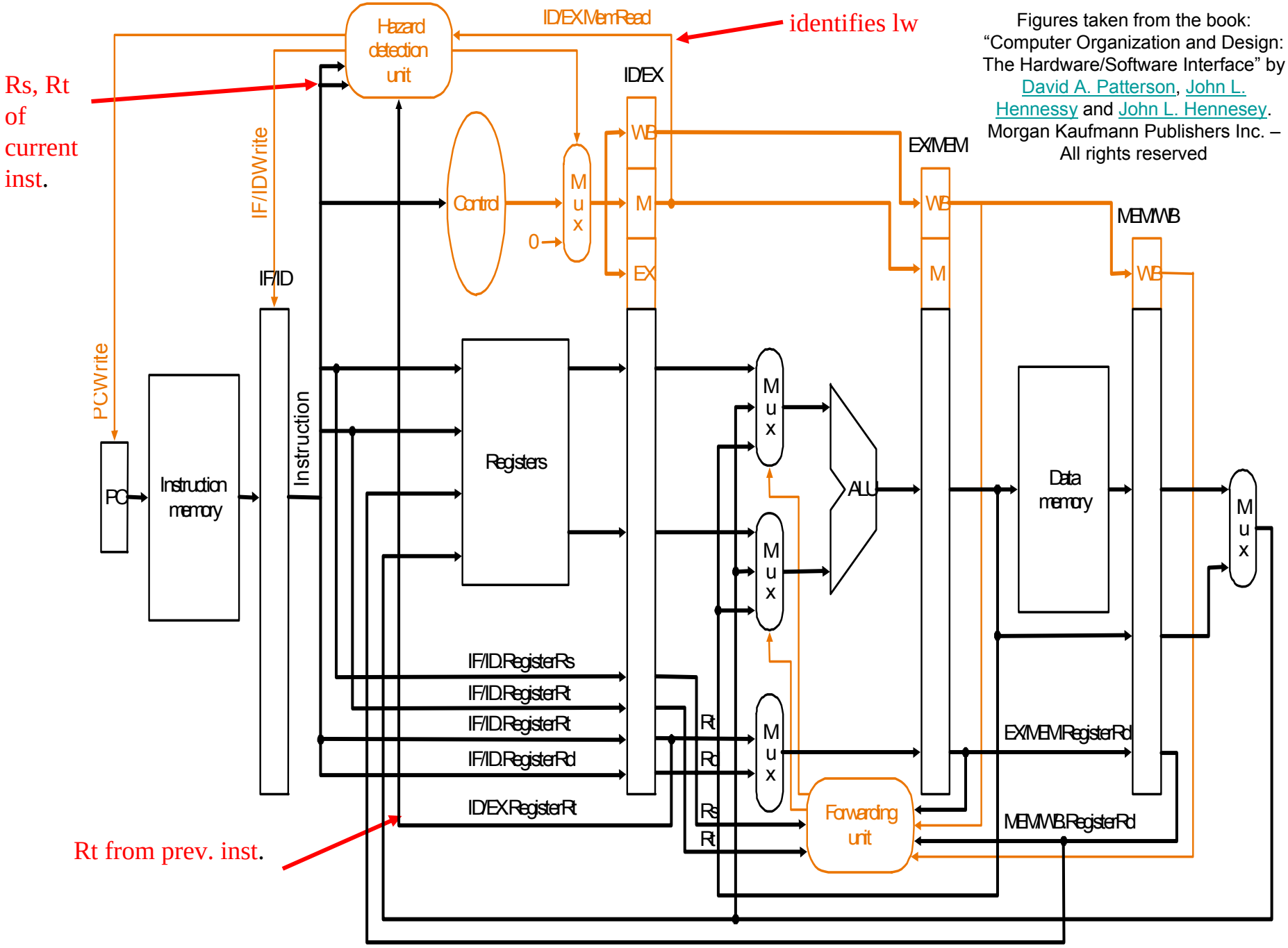
Actual Instruction and not a FLUSH

# פתרון ב - הערות

- כדי שפתרון ב' יעבוד עלינו לשנות מעט את החומרה:
  - יש לבדוק אם אוגר ה-rd של הפקודה sw זהה לאוגר היעד של הפקודה lw(כמו בפתרון הקודם).
  - אבל, אוגר ה-rd אינו זמין בשלב ה-MEM היות והכניסה ל-MUX ה-RegDst היא X (don't care)
    *(כזכור RegDst בוחר בין rd,rt כאוגר היעד של פקודה)*
  - היות וערך *RegDst* אינו בשימוש sw ניתן לבחור לו שימוש ולכן נבחר שערכו יהיה כך שייבחר ב-MUX האוגר rt.
  - זה מבטיח שאוגר המקור של sw יהיה זמין עבור ההתניות שבשקף הקודם (rd יכיל את rt של פקודת sw ב-EX/MEM)

# Control





# הערות

- *נמנע מ-stall רק אם אוגר ההיסט (rs) של sw אינו אוגר היעד של lw.*

```
if ID/EX.MemRead and                                    // lw in EX stage?
        ((ID/EX.RegisterRt = IF/ID.RegisterRs) or       // same register?
        (ID/EX.RegisterRt = IF/ID.RegisterRt)) and      // but not...
        not (IF/ID.MemWrite and                         // sw in ID stage?
        (ID/EX.RegisterRt = IF/ID.RegisterRs)) and      // same register?
        (ID/EX.RegisterRt <> 0)                         // register r0?
then
        Stall the pipeline
```

- IF/ID.MemWrite הינו סיגנל חדש שמסמן פקודת store ואשר יפוענח מה-opcode.

# דוגמא

- סדרו את פקודות התכנית הבאה על-מנת שמספר מחזורי השעון שתצטרך יהיה המקסימלי (הכי גרוע מבחינת ביצועים) תוך מתן אותה תוצאה

```
lw      $3, 0($5)
lw      $4, 4($5)
add     $7, $7, $3
add     $8, $8, $4
add     $10, $7, $8
sw      $6, 0($5)
beq     $10, $11, Loop
```

- עלינו להשתדל שיהיו כמה שיותר stalls

```
lw      $3, 0($5)
add     $7, $7, $3      # requires stall on $3
sw      $6, 0($5)
lw      $4, 4($5)
add     $8, $8, $4      # requires stall on $4
add     $10, $7, $8
beq     $10, $11, Loop
```