BAR-ILAN UNIVERSITY

School of Engineering

אוניברסיטת בר-אילן

בית הספר להנדסה

מבוא להנדסת מחשבים - פתרון מקוצר

תשע"ד סמס' א' מועד ב'

83-252

- מרצה : פרופ' שמואל וימר
- מתרגלת : אילת חיימוביץ

- **חומר עזר מותר:** מחברות/ דפים מהרצאות, תרגולים ותרגילי בית, ספר computer organization and design.
- **משך המבחן שלוש שעות**

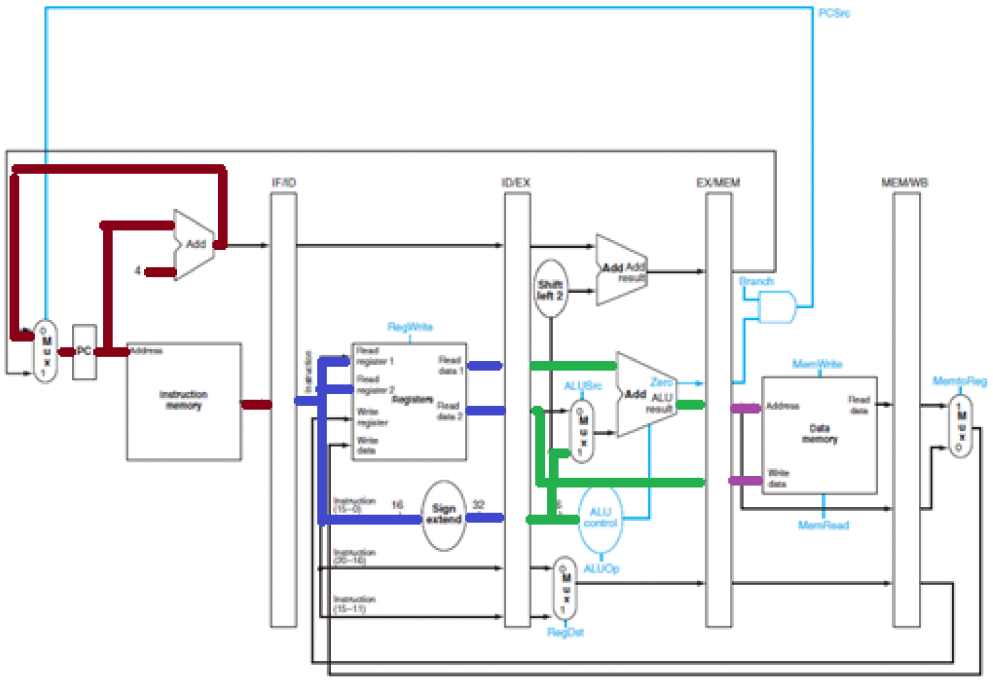
- **סך כל הנקודות הוא 110**.
- משקל השאלות השונות נתון בגוף השאלות.
- **יש לנמק את כל תשובותיכם**. אין צורך לפתח מחדש תוצאות שהוכחו בכיתה, אלא אם כן נאמר מפורשות לעשות כן.
- **יש לשרטט דיאגראמות באופן ברור !**

בהצלחה

שאלה מס' 1 (20 נק')

עבור פקודת SW נדרש לסמן את קווי הנתונים הפעילים בכל אחד משלבי הפקודה השונים. **יש להשתמש בדפים המצורפים לטופס הבחינה, ולסמן כל שלב בדף נפרד.**

פתרון: שימו לב - כל שלב מסומן בצבע שונה.



שאלה מס' 2 (35 נק')

נתונה מכונת ה MIPS שלהלן. מתבצע קטע הקוד הבא:

```
add $3, $2, $1
sub $4, $3, $5
add $5, $3, $7
add $6, $7, $1
add $8, $2, $6
```

בהמשכו הוכנסה סדרה של חמש פקודות NOP.

א. אילו אוגרים נקראים ואילו אוגרים נכתבים במחזור שעון 3, 4, 5, 6 ו 7?
יש לציין את שמות האוגרים הנכתבים והנקראים לחוד, ולכל מחזור לחוד.

מחזור שעון שלישי: הפקודה השנייה (sub $4, $3, $5) נמצאת בשלב ID, על כן נקראים רגיסטרים $3 ו- $5.
במחזור שעון זה אין פקודה בשלב WB ולא נכתב מידע באף רגיסטר.

מחזור שעון רביעי: הפקודה השלישית (add $5, $3, $7) נמצאת בשלב ID, על כן נקראים רגיסטרים $3 ו- $7.
במחזור שעון זה אין פקודה בשלב WB ולא נכתב מידע באף רגיסטר.

מחזור שעון חמישי: הפקודה add $6, $7, $1 נמצאת בשלב ID, על כן נקראים רגיסטרים $1 ו- $7.
במחזור שעון זה הפקודה הראשונה נמצאת בשלב ה- WB ועל כן נכתב רגיסטר $3.

מחזור שעון שישי: הרגיסטרים הנקראים הם $6, $2. הרגיסטר הנכתב הוא $4.

מחזור שעון שביעי: הרגיסטר הנקרא הוא רגיסטר האפס (הכתובת 00000 נמצאת בכניסות Read Register 1 ו- Read Register 2). הרגיסטר הנכתב הוא $5.

ב. כנ"ל למחזור 9, 10 ו 11.

למעשה במחזורים 9-11 נקרא רגיסטר האפס. במחזור 9 נכתב רגיסטר $8, ובמחזורים 10-11 נכתב רגיסטר האפס (שימו לב שנכתב הערך 0 לרגיסטר האפס).

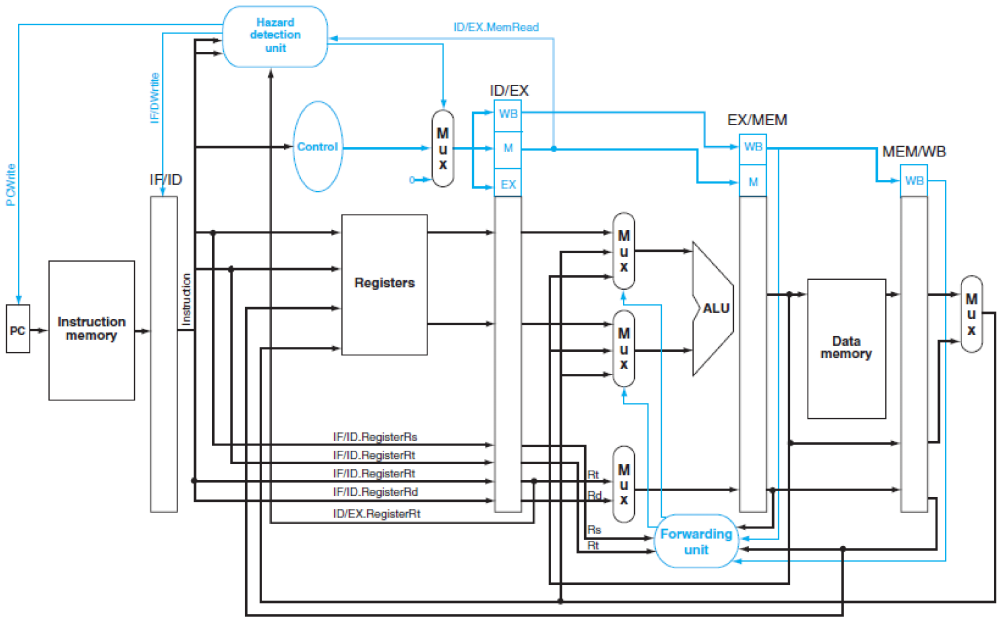
ג. מה מבצעת יחידת ה FORWARDING במחזור 5? במידה ומתבצעות השוואות כלשהן, ציין מהן.

יחידת ה FORWARDING בודקת אם הרגיסטרים הנקראים בפקודה add $5, $3, $7 מעודכנים. מכיוון שמדובר ב- pipelined MIPS יתכן שאחת או שתי הפקודות הקודמות אמורות לשנות את ערך הרגיסטרים הנקראים, אך עוד לא הספיקה לכתוב את הערך החדש ב- register file. לכן יש להשוות את 2 כתובות הרגיסטרים הנקראים לכתובת הרגיסטרים הנכתבים ב-2 הפקודות הקודמות. יחידת ה FORWARDING שולטת על הכניסות ל-ALU ומוודאת שהערך העדכני של הרגיסטרים הוא הערך בו נשתמש בחישוב.

ההשוואות הינן:
4=3 ?
4=7 ?
3=3 ?
3=7 ?

ד. כנ"ל למחזור 6.
ההשוואות הינן:

7=5 ?
1=5 ?
7=4 ?
1=4 ?



<u>שאלה מס' 3 (35 נק')</u>

במכונת MIPS מרובת מחזורים רוצים לממש בפקודה אחת הנקראת **I_inc** שתפקידה לטעון מילה מהזיכרון ולהגדיל ב 1 את אוגר האינדקס (הבסיס). **I_inc** מממשת בפועל את שתי הפקודות שלהלן:

```
lw $rs, L($rt)

addi $rt, $rt, 4
```

נתונות סכמת ה MIPS ומערכת הבקרה שלהלן.

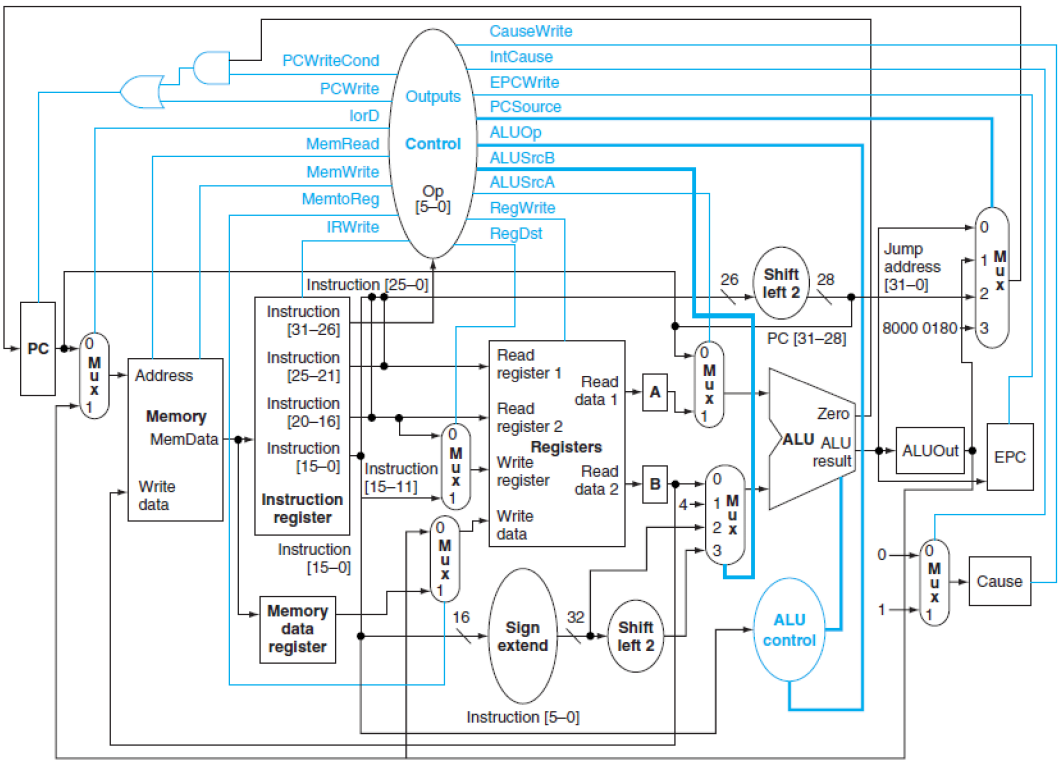
א. נדרש פתרון בו השינויים הינם במערכת הבקרה בלבד.
   a. עדכן את מכונת המצבים בהתאם על גבי הטופס המצורף.
   b. האם נדרשים אותות בקרה נוספים? נמק בפירוט.
   c. מה משמעות השינוי מבחינת ביצועי המערכת לעומת מימוש בשתי פקודות נפרדות?

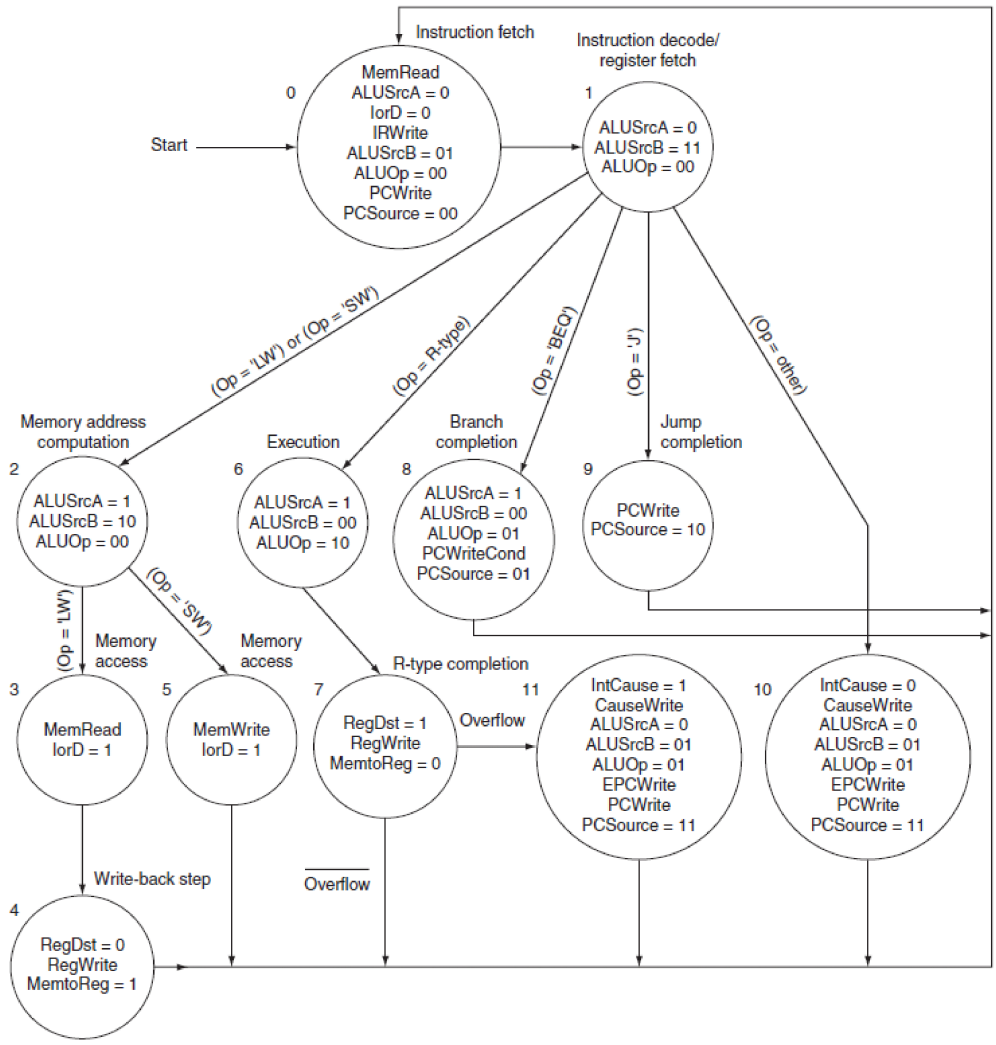
נוסיף 2 מיקרו פקודות חדשות.
מיקרו פקודה 12 תדאג לחישוב ערך אוגר האינדקס החדש: ALUOp=00, ALUSrcA=1, ALUSrc=01
לאחר מיקרו פקודה זו ישמר ב- ALUOut הכתובת אותה יש לכתוב ל- register file. על כן מיקרו פקודה 13 תהיה: RegDst=10 ,MemToReg=0, RegWrite. סדר מיקרו פקודות

לפקודה החדשה יהיו 0->1->2->3->4->12->13 (לא לשכוח להוסיף מעבר של I_inc ממצב 4 ל-12, ולציין שהמעבר ממצב 4 ל-0 יתקיים תחת opcode של LW).
לא הוספנו אותות בקרה נוספים. למעשה הפקודה החדשה כוללת את כל מיקרו הפקודות של LW (as is), ועל כן מובן שלא צריך להוסיף אותות בקרה לביצוע פעולה שהמעבד יודע לבצע. לאחר סיום 5 מיקרו פקודות של lw נותר לנו לחשב קידום של הערך ב- Read Data 1 ב-4 ולכתוב ב- register file. ב- mux המזין את הכניסה העליונה של ה- ALU ישנה אפשרות להעביר את Read Data 1 (ALUSrcA=1), ול- mux המזין את הכניסה התחתונה של ה-ALU ישנה אפשרות להעביר 4 (ALUSrcB=01), ועל כן חישוב הקידום לא מצריך אותות בקרה נוספים. פקודות R-type (בין היתר) מצריכות כתיבת חישוב ב-ALU ל- Register file, וה- mux שמזין את Write Register ישנה אפשרות להעביר את כתובת רגיסטר $Rt (RegDst=10), ועל כן אין צורך בתוספת אותות בקרה עבור הפקודה.
ביצוע ב-2 פקודות יצריך 9 מחזורי שעון (5 עבור lw ו-4 עבור addi). המימוש שלנו יצריך 7 מחזורי שעון, כלומר חסכנו 2 מחזורי שעון במימוש בפקודה אחת.

ב. כעת ניתן להוסיף יחידה אריתמטית וגם לשנות את ה REGISTER FILE.
   a. עדכן את מכונת המצבים בהתאם על גבי הטופס המצורף.
   b. האם נדרשים אותות בקרה נוספים? נמק בפירוט.
   c. מה משמעות השינוי מבחינת ביצועי המערכת לעומת המימוש בסעיף א'?

נוסיף יחידה אריתמטית המבצעת חיבור בלבד (בדיוק כפי שעשינו עבור single cycle MIPS ו-pipelined MIPS - נחווט את ה-ALU כך שתמיד יבצע חיבור), ואליה יכנס Read Data 1 ו- 4. מוצא ה- ALU החדש יכנס לפורט כתיבה חדש (Write Data 2) שנוסיף ל- register file. פורט הכתיבה החדש נשלט על ידי אות RegWrite2. לא נצטרך להוסיף פורט Write Register 2 מכיוון שהפורט החדש תמיד כותב לכתובת שבכניסה Read Register 1.
כעת במחזור השעון השלישי/רביעי/חמישי של הפקודה ניתן כבר לכתוב את קידום אוגר האינדקס - נוסיף מיקרו פקודה יחידה 12, אשר תשכפל את מיקרו פקודה 2/3/4 ותוסיף RegWrite2. המעבר ממיקרו פקודה זו הינה למיקרו פקודה 3/4/0 בהתאמה.
כאמור נוסיף אות בקרה RegWrite2 שישלוט על פעולת הכתיבה לרגיסטר Read Register 1 (Rt). בגלל תוספת ה-ALU אות בקרה זה הוא היחיד הדרוש לקידום. עבור הקריאה מהזיכרון והכתיבה ל- register file נשתמש באותם ערכים בדיוק לאותות הבקרה כמו עבור הפקודה LW.
כעת הפקודה מתבצעת ב-5 מחזורי שעון במקום 7. חזרנו למעבד בו הפקודה הארוכה ביותר היא 5 מחזורי שעון.


CauseWrite
IntCause
EPCWrite
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5–0]
PC
0
M
u
x
1
Address
Memory
MemData
Write
data
Instruction [25–0]
Instruction
[31–26]
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–0]
Memory
data
register
Instruction
[15–11]
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
16
Sign
extend
32
Shift
left 2
Instruction [5–0]
26
Shift
left 2
28
PC [31–28]
Jump
address
[31–0]
8000 0180
0
1
2
3
M
u
x
Zero
ALU
ALU
result
ALUOut
EPC
ALU
control
Cause



שאלה מס' 4 (20 נק')

לפניך קוד לולאה שבו $10 מאותחל ל 0 ו $30 מאותחל ל 400. הקוד רץ על מכונת PIPELINED MIPS סטנדרטית המטפלת ב DATA HAZARDS. שים לב שהטפול בקפיצה מתבצע בשלב MEM.

א. בכמה מחזורי שעון יתבצע הקוד הנ"ל?

הפקודות בלולאה מבצעות 50 פעמים (שימו לב שבכל מעבר בלולאה יש קידום ב-8 של $10). לאחר כל פקודת lw יש השהייה של מחזור שעון אחד (הפקודה הראשונה כותבת לרגיסטר $2 במחזור שעון 5, הפקודה שאחריה משתמשת בערך $2 במחזור שעון 4. שימו לב שבתחילת מחזור 4 עוד לא בוצע קריאה מהזיכרון והערך העדכני של $2 לא זמין, בפקודת lw יש אותה בעיה ביחס לרגיסטר $5).
בהנחה שהמעבד משתמש בשיטת הטיפול בקפיצות מותנות הראשונה שנלמדה בהרצאה (הוספת 3 nop) אזי לכל מעבר בלולאה נדרשים 13 מחזורי שעון, סך הכל: 13*50 מחזורי שעון.

ב. האם ניתן לשנות את סדר הפקודות כך שהלולאה תתבצע מהר יותר, ומבלי שהתוצאות ישתנו? במידה וכן, הסבר כיצד ורשום את הקוד החדש.
לא ניתן לשנות את סדר הפקודות כדי לפתור את ה- branch hazard, אבל ניתן לפתור את ה- load hazards על ידי שינוי סדר הפקודות כך שהפקודה שאחרי lw לא תצטרך לקרוא את הערך הנקרא מהזיכרון. כמובן שכדי לוודא שהתוצאות לא ישתנו לא נעביר פקודה שקוראת ערך של רגיסטר X לפני פקודה שכותבת לאותו רגיסטר (למשל לא נשנה את ערך sw ו- addi).

```
lw $2, 0($10)
lw $5, 4($10)
sub($4, $2, $3)
sw $4, $2, $3
sub $6, $5, $3
sw $6, 4($10)
addi $10, $10, 8
bne $10, $30, Loop
```

ג. כמה מחזורי שעון ייקח הקוד החדש? חסכנו 2 מחזורי שעון בכל מעבר בלולאה, ועל כן הקוד ייקח 11*50 מחזורי שעון.

```
Loop:   lw   $2,  0($10)
        sub  $4,  $2, $3
        sw   $4,  0($10)
        lw   $5,  4($10)
        sub  $6,  $5, $3
        sw   $6,  4($10)
        addi $10, $10, 8
        bne  $10, $30, Loop
```