אוניברסיטת בר-אילן

הפקולטה להנדסה

BAR-ILAN UNIVERSITY

Engineering Faculty

תכן לוגי ומבוא למחשבים

תשע"ז סמס' א' מועד ב'

83-253

- מרצה : פרופ' שמואל וימר
- מתרגלים : גב ספיר ארליך, מר יוסף לונדון, מר איתמר לוי
- **חומר עזר מותר:** שקפי ההרצאות כולל הערות והסברים שנרשמו במהלך ההרצאות וספר הקורס.
- משך המבחן:

  שלוש שעות.
- חובה לענות על כל השאלון. סה"כ נקודות אפשרי 110, הציון הסופי לא יעלה בכל מקרה על 100.
- **יש לנמק את כל תשובותיכם**. אין צורך לפתח מחדש תוצאות שהוכחו בכיתה, אלא אם כן נאמר מפורשות לעשות כן.
- יש לשרטט דיאגראמות באופן ברור !
- הכתיבה בעט בלבד. כתיבה בעפרון לא תיבדק.

בהצלחה!

<u>שאלה א'</u> (35 נקודות)

נתון מעבד MIPS בעל מחזור יחיד כמופיע בתרשימים 1, 2 ו 3 (שלשתם זההים). בכל השאלות הבאות נדרש להרחיב את סט הפקודות וזאת ע"י שינויים מזעריים ככל האפשר במעבד הקיים. במדה ונדרש הוסיפו לתרשימים המתאימים משאבי חומרה ואותות בקרה, וקיבעו את ערכיהם של אותות הבקרה הקיימים ונוספים במדה שנדרש בטבלאות המתאימות. הסבירו באופן מלא ומדוייק את פתרונכם.

1. (10 נק') נדרש להוסיף תמיכה בפקודה חדשה **LWR $rd, $rs, $rt** המשתמשת ב R-format. הפקודה מחשבת כתובת אפקטיבית ע"י סיכום הרגיסטרים $rs ו $rt .

	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0	Jump	
R-format	1	0	0	1	0	0	0	1	0	0	
lw	0	1	1	1	1	0	0	0	0	0	
sw	x	1	x	0	0	1	0	0	0	0	
beq	x	0	x	0	0	0	1	0	1	0	
J	x	x	x	0	0	0	x	x	x	1	
LWR											

תשובה: לא נדרשת שום תוספת חומרה שהיא ושום אותות בקרה חדשים (תרשים 1 נשאר ללא שינוי). קביעת אותות הבקרה הקיימים עבור פקודת LWR מופיעה להלן.

	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0	Jump	
R-format	1	0	0	1	0	0	0	1	0	0	
lw	0	1	1	1	1	0	0	0	0	0	
sw	x	1	x	0	0	1	0	0	0	0	
beq	x	0	x	0	0	0	1	0	1	0	
J	x	x	x	0	0	0	x	x	x	1	
LWR	1	0	1	1	1	0	0	0	0	0	

2. (10 נק') נדרש להוסיף תמיכה בפקודה קפיצה חדשה **jm offset ($rs)** המשתמשת ב I-format. הפקודה החדשה קוראת מהזכרון את הערך המצוי בכתובת האפקטיבית המוגדרת בפקודה וגורמת לתכנית המבוצעת לקפוץ לכתובת הנ"ל.

	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0	Jump	
R-format	1	0	0	1	0	0	0	1	0	0	
lw	0	1	1	1	1	0	0	0	0	0	
sw	x	1	x	0	0	1	0	0	0	0	
beq	x	0	x	0	0	0	1	0	1	0	
J	x	x	x	0	0	0	x	x	x	1	
jm											

תשובה: נדרשת הרחבת ה MUX המשמש קפיצות בכניסה שלישית, דבר המחייב שתי סיביות באות הבקרה Jump. את הכניסה הנוספת של ה MUX מחברים ליציאת זכרון הנתונים.

	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0	Jump	
R-format	1	0	0	1	0	0	0	1	0	00	
lw	0	1	1	1	1	0	0	0	0	00	
sw	x	1	x	0	0	1	0	0	0	00	
beq	x	0	x	0	0	0	1	0	1	00	
J	x	x	x	0	0	0	x	x	x	01	
jm	x	1	x	0	1	0	x	0	0	10	

3. (2 נק') איזה יתרון יש לפקודה על פני jump רגיל?

תשובה: יתרונה של הפקודה בכך שהיא מאפשרת קפיצה לכל כתובת שהיא, בעוד ש jump רגיל מספק רק 28 סיביות, וארבחת ה MSB מוכתבים ע"י ה PC.

4. (10 נק') נדרש להוסיף תמיכה בפקודה קפיצה חדשה **jal** (jump and link) המשתמשת לקפיצה לסברוטינה, כך שבסיומה ידוע לאן צריך לחזור בתכנית הקוראת בכדי להמשיך בבצוע התכנית. כתובת החזרה נשמרת ב $ra ($31).

	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0	Jump	
R-format	1	0	0	1	0	0	0	1	0	0	
lw	0	1	1	1	1	0	0	0	0	0	
sw	x	1	x	0	0	1	0	0	0	0	
beq	x	0	x	0	0	0	1	0	1	0	
J	x	x	x	0	0	0	x	x	x	1	
JAL											

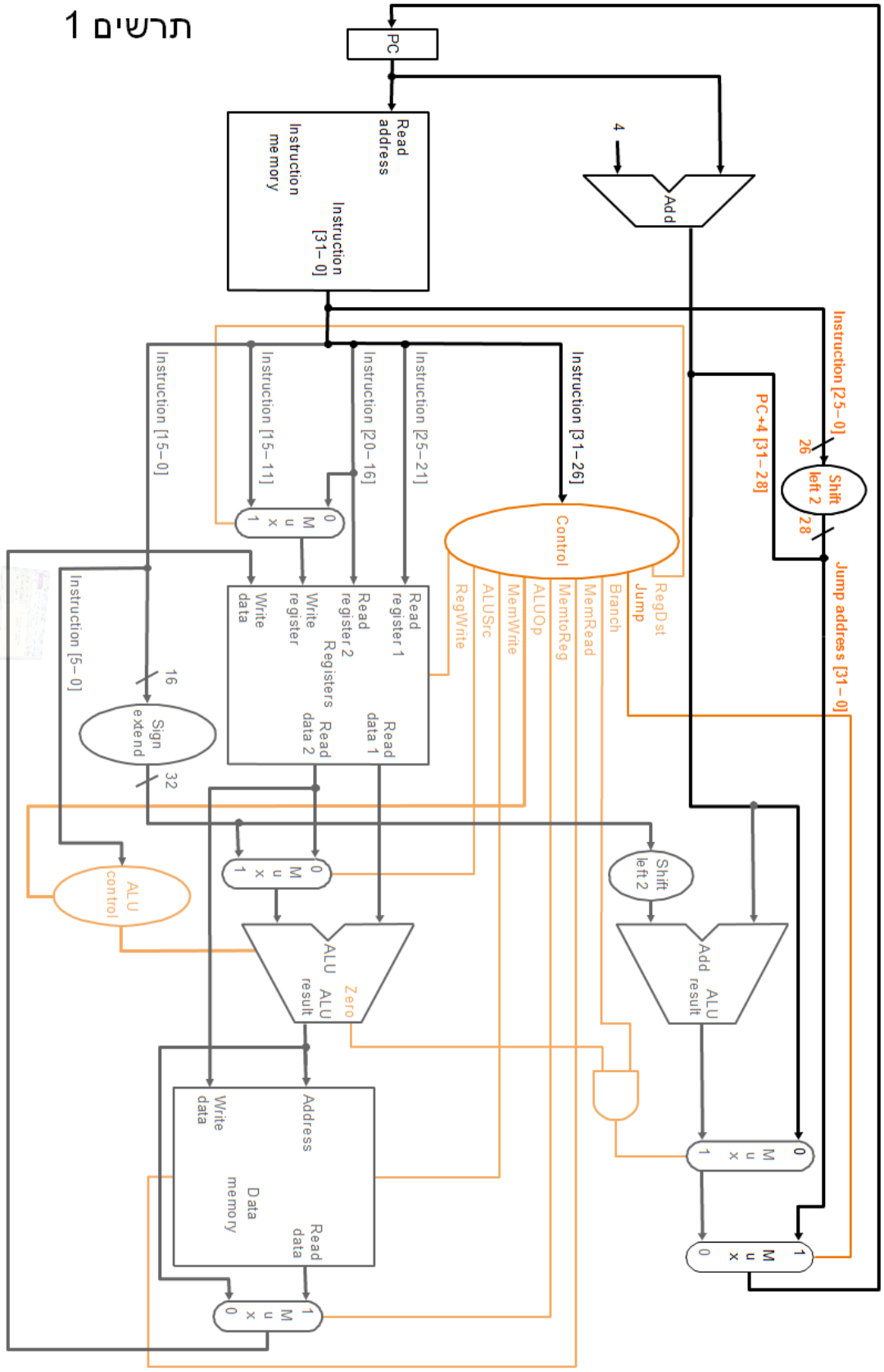
תשובה: נדרשת הרחבת ה MUX הנשלט על ידי אות הבקרה RegDst לכניסה שלישית של הערך הקבוע31, דבר המחייב שתי סיביות באות הבקרה. כמו כן, מאחר ולאוגר ($31) $ra נדרש להעביר את כתובת החזרה מהסברוטינה, יש להוסיף ל MUX הנשלט ע"י MemtoReg כניסה נוספת של PC+4 . אותות הבקרה של שני ה MUX הנ"ל הופכים לשתי סיביות.

	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0	Jump	
R-format	01	0	00	1	0	0	0	1	0	0	
lw	00	1	01	1	1	0	0	0	0	0	
sw	xx	1	xx	0	0	1	0	0	0	0	
beq	xx	0	xx	0	0	0	1	0	1	0	
J	xx	x	xx	0	0	0	x	x	x	1	
JAL	10	x	10	1	0	0	x	x	x	1	

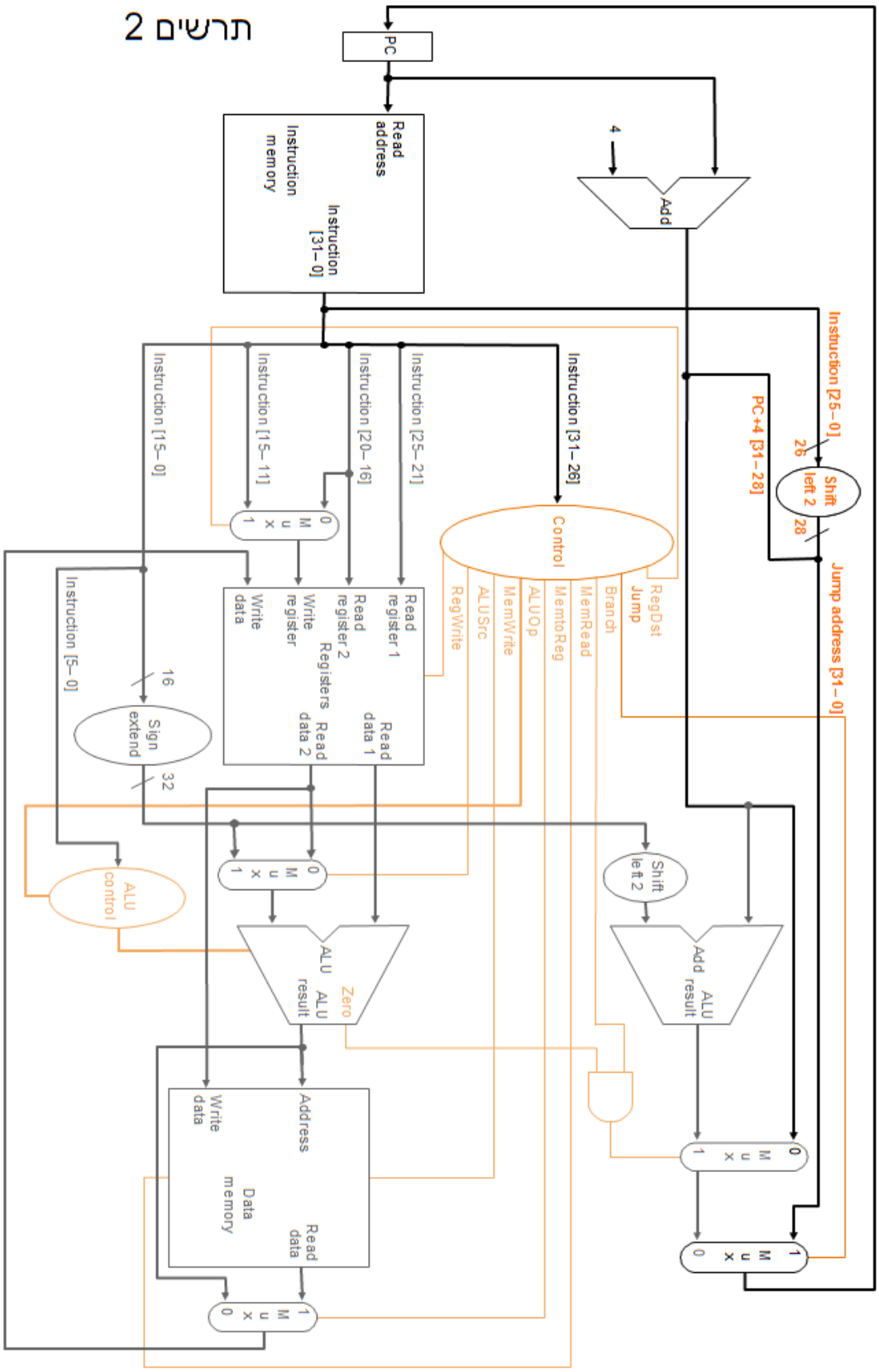
5. (3 נק') פקודת **return** גורמת לחזרה מהסברוטינה. הסבירו האם נדרשת תוספת חומרה למעבד, ובמדה ונדרש, מה התוספת (אין צורך לצייר)?

תשובה: מאחר ונדרשת מהסברוטינה חזרה לכתובת שנשמרה באוגר ($31) $ra, יש ראשית לקרוא את האוגר הזה. יש אם כך להעביר את הערך 31 לאחת מכניסות הקריאה של ה register file, ואת היציאה המתאימה לנתב ל PC. הדבר יחייב MUX חדש באחת מכניסות הקריאה של ה register file, ורחבת ה MUX השולט על ה PC.

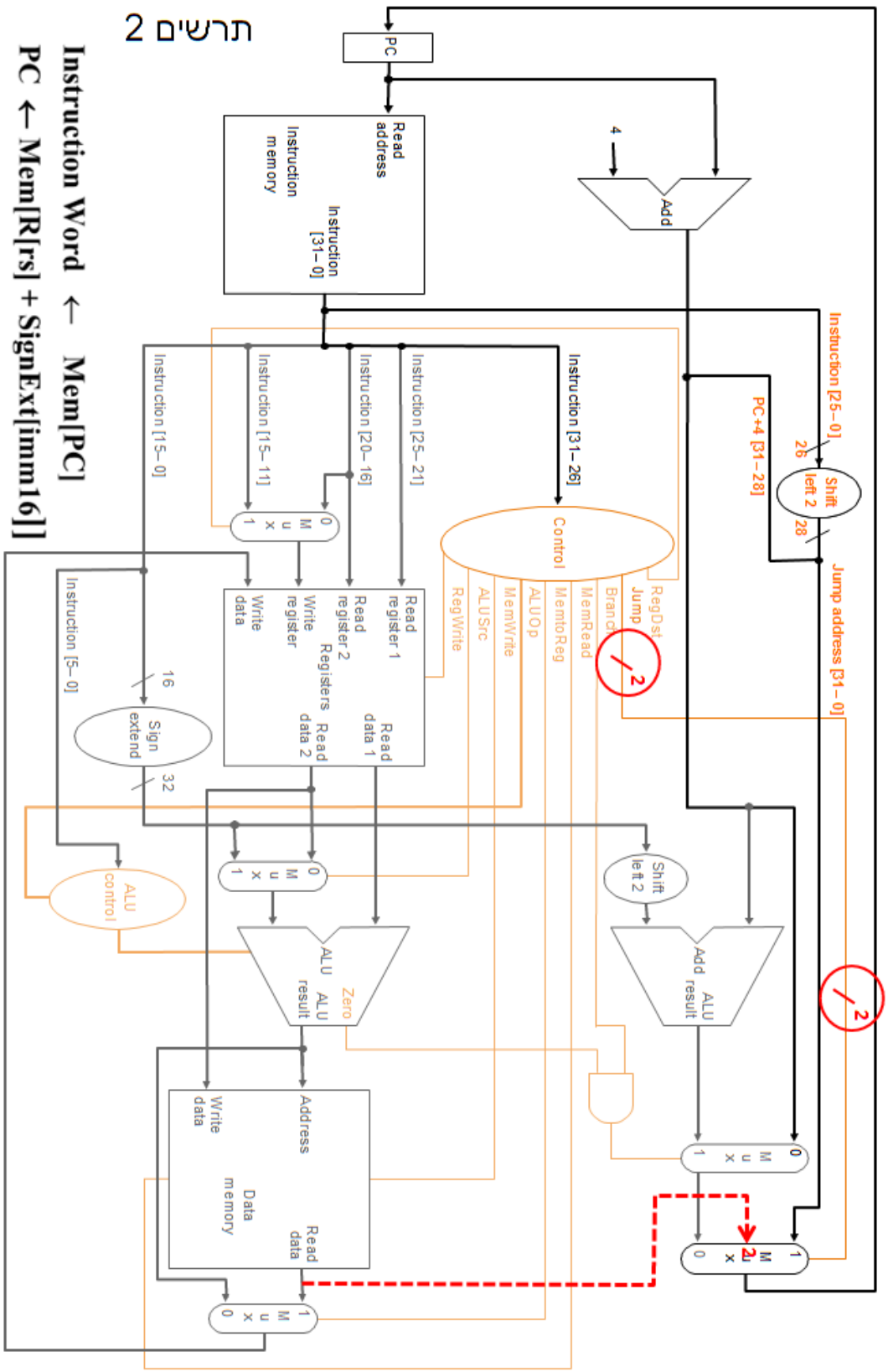
תרשים 1


PC
Read address
Instruction [31–0]
Instruction memory
4
Add
Instruction [25–0]
26
Shift left 2
28
PC+4 [31–28]
Jump address [31–0]
Instruction [31–26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Control
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
M u x
0
1
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
ALU control
ALU
Zero
ALU result
Shift left 2
Add
Address
Read data
Data memory
Write data

תרשים 2


PC
Read address
Instruction [31–0]
Instruction memory
4
Add
Instruction [25–0]
26
Shift left 2
28
PC+4 [31–28]
Jump address [31–0]
Instruction [31–26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
M u x
0
1
Control
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
ALU control
ALU
Zero
ALU result
Shift left 2
Add
Address
Read data
Data memory
Write data

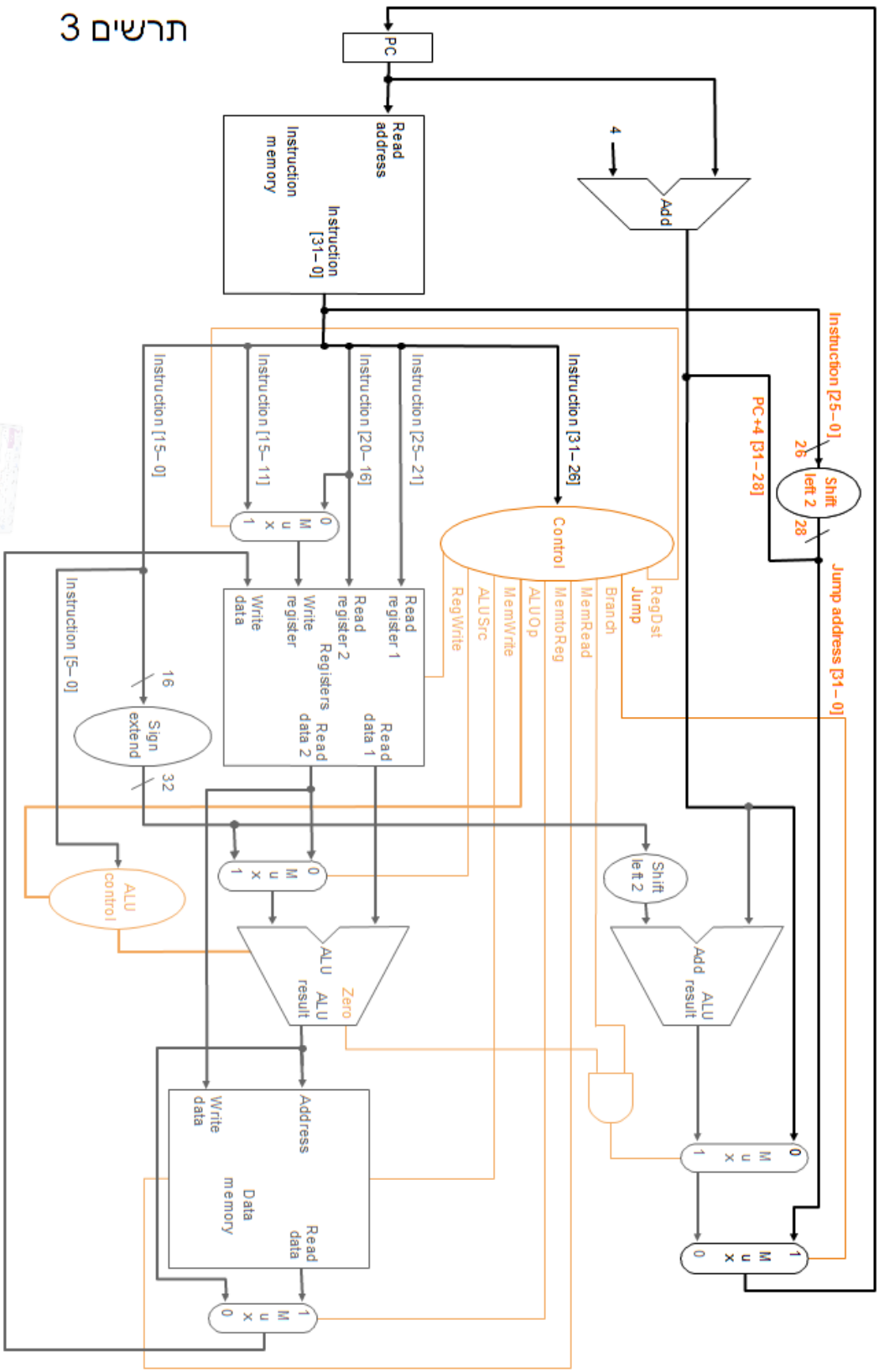
תרשים 2



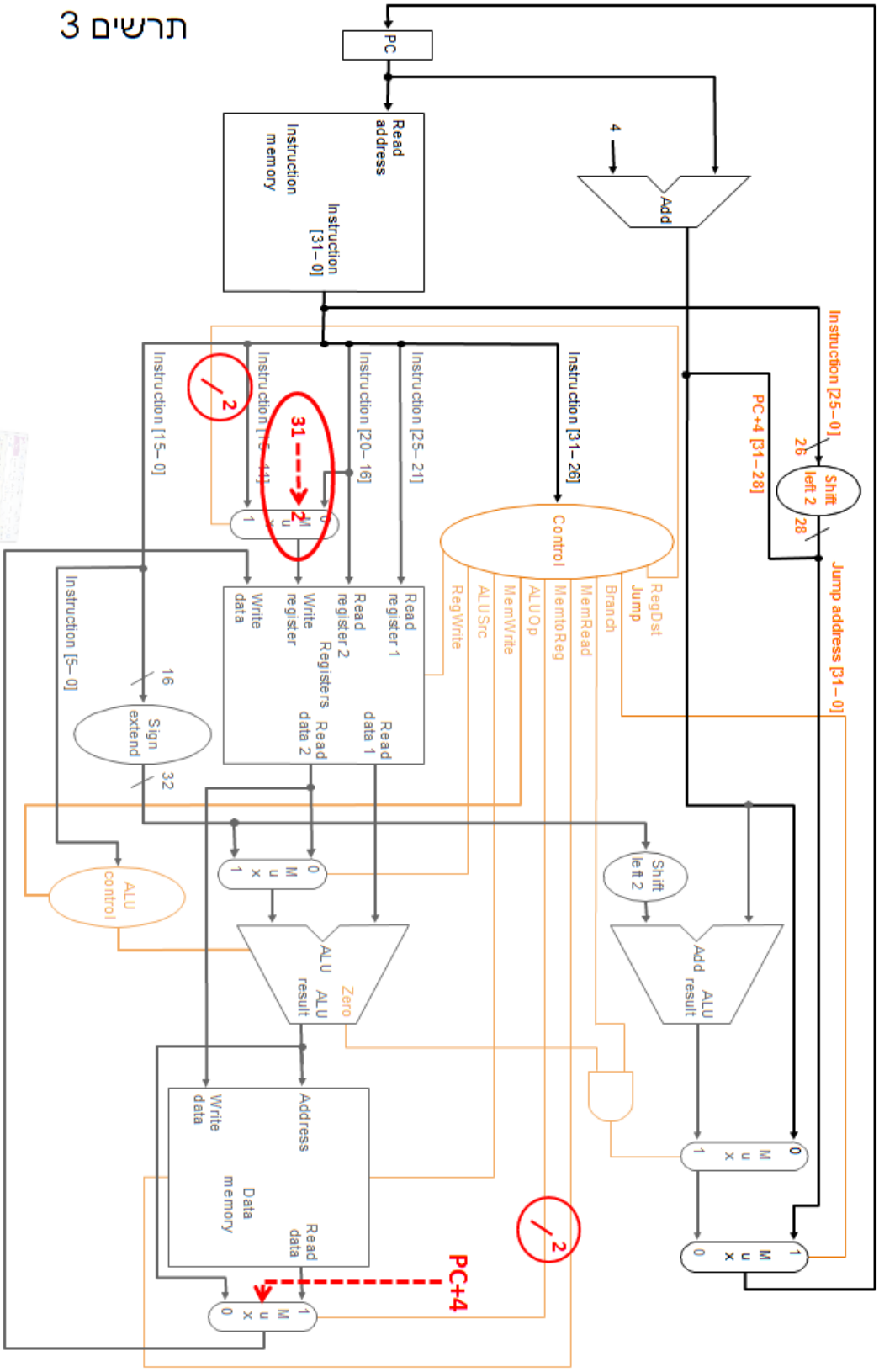
Instruction Word ← Mem[PC]

PC ← Mem[R[rs] + SignExt[imm16]]

תרשים 3


PC
Read address
Instruction [31–0]
Instruction memory
4
Add
Instruction [25–0]
26
Shift left 2
28
PC+4 [31–28]
Jump address [31–0]
Instruction [31–26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Control
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
16
Sign extend
32
ALU control
Shift left 2
Add
ALU result
ALU
Zero
ALU result
Address
Write data
Data memory
Read data
M u x

תרשים 3


PC
Read address
Instruction [31–0]
Instruction memory
4
Add
Instruction [25–0]
26
Shift left 2
28
PC+4 [31–28]
Jump address [31–0]
Instruction [31–26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
2
31
M u x
Control
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
ALU control
Shift left 2
Add
ALU result
ALU
Zero
Address
Data memory
Read data
Write data
PC+4

<u>שאלה ב (45 נקודות)</u>

בשאלה זו אתם נדרשים לממש את הפקודה SWAP המחליפה בין ערכי שני רגיסטרים הנמצאים בקובץ הרגיסטרים. המערכת תמומש במעבד PIPELINE MIPS הנמצאת בתרשים XXX.

א. (5 נקודות) ללא הפקודה החדשה, בקונפיגורציה הזאת, מהם ההשוואות שנעשות ביחידת הFORWARDING בשלב הDECODE? רשמו **במדויק** ופרטו **בקצרה** למה נצרך כל השוואה. האם יש השוואה לצורך פתרון בעיית LOAD HAZARD?

- בתרגול נתבקשתם במפורש לפתור את הסעיף הזה בבית מספר פעמים.
- Detect hazard conditions:
  - 1a) ID/EX.Rd = IF/ID.Rs
  - 1b) ID/EX.Rd = IF/ID.Rt
  - 2a)EX / MEM.Rd = IF/ID.Rs
  - 2b) EX/ MEM.Rd = IF/ID.Rt
  - (also check for RegWrite, zero, and newest)
- Use these checks to control the forwarding multiplexers

LOADHAZARD נפתר ע"י מערכת אחרת.

FORWARDING החדש בשלב הDECODE מיועד רק לפקודות BRANCH ואינו מחליף ערכים רק בוחן השוואה

כל מה שקורה בשלב הWB מספיק להכתב ולהקרא נכון כפי שנלמד בכיתה והודגש בתרגול.

ב. (10 נקודות) אתם נדרשים לממש את הפקודה ע"י שינויים מינמליים. מותר להגדיל בוררים (mux) קיימים, ולהוסיף בורר יחיד נוסף, לספק כניסות קבועות נוספות, וכן שינויים מינימליים בבקר (הוצאה והכנסה של קוי בקרה וזיהוי opcode חדש). כמובן שבהקשר הנל מותר גם כן להוסיף סיביות לאותות הבקרה שיאגרו באוגרי הצינור. ממשו. בחלק זה לא להתייחס ליחידות הFORWARDING השונים והHAZARD UNIT.

בתרגול נאמר לכם במפורש כיצד לפתור שאלה מעין זה. ראשית זיהוי מה נצרך (קריאה משני רגיסטרים וכתיבה לשני רגיסטרים), זיהוי הבעיות (לא ניתן לכתוב לשני רגיסטרים בו זמנית. לא ניתן לכתוב בצורה טורית בפקודה מעין זאת שכן המידע נדרס) זיהוי עקרון לפתרון (מתוך מגבלות השאלה נשאר אפשרות לבצע ריבוב בזמן, כלומר למרוח את הפקודה על פני שני פקודות בדומה לmulti cycle mips) ניסיון מימוש וטיפול בבעיות נלוות (HAZARD UNIT וכדו). הפתרון המוצע כאן מבוסס לחלוטין על הפתרון שנלמד בכיתה לLoad Hazard. יצירת עצירה של התקדמות המערכת ושמירת המידע בדרך.

יתרה מזאת, הוזכר בתרגול ונתבקשתם בתרגול לפתור שאלה זו בעצמכם.

אסור לבצע שינויים בקובץ הרגיסטרים. רק השינויים שכתובים בשאלה היו מותרים. כפי שהוסבר פעמים רבות בתרגול. כמו כן אסור לייצר פקודה שסתור פקודה אחרת. על שני דברים אלו יורדו הרבה נקודות.

נבצע זאת בשני שלבים. הפקודה החדשה SWAP תהיה בעלת OPCODE מסוים חדש. כמו כן, הקומפיילר אוטומטית תכניס את הכתובת של הרגיסטר השני לתוך כתובת רגיסטר היעד. בשלב הראשון של הפקודה הפקודה תקרא את שני הרגיסטרים ותמשיך הלאה כפקודה רגילה. כאשר **הבקר** יגלה שמדובר בפקודת SWAP (שלב הDECODE) הוא יבצע **העתקה** של כל המידע שברגיסטר IF/ID מלבד הOPCODE לתוך **עצמו ע"י הבורר החדש**. כמובן שגם לא יתן לPC להיכתב מחדש. לתוך סיביות הOPCODE של הרגיסטר הנל ייכנסו OPCODE חדש SWAP2. עקב כך, במחזור שעון הבא הכל יקרה שוב אלא שהפעם המערכת

תדע כי מדובר בחלק השני של פקודת הSWAP שכן יש OPCODE אחר. במחזור שעון זה, החלק הראשון של הפקודה נמצאת בשלב הEX. כאן ייסכם הרגיסטר עם רגיסטר האפס וימשיך במורד הPIPE כפקודת RTYPE רגילה. הדבר נעשה ע"י שליטה בבוררי הכניסה לALU ע"י סיבית בקרה נוספת (כל המצבים 5והלאה של הבורר יעבירו את ה0 או לחילופין לוגיקת בורר מורכבת יותר של אם הסיבית החדשה דלוקה ה0 עובר).

עבור השלב השני של פקודת הSWAP נדרש להעביר גם כן את יעד הכתיבה של הרגיסטר השני שהוא כתובת הרגיסטר הראשון. כתובת זו מועברת בPIPE גם ככה לצורך מערכת הFORWARDING של שלב הEX. ניקח אותה משם ונכניסה גם כן לבורר היעד שיש בשלב הEX ונרחיב את קו הבקרה שלו ל2 סיביות. בזה תמו השינויים מלבד עדכון מערכת הFORWARDING של שלב הEX.

וריאציה של אותו עיקרון – ביצוע אותה פקודת RTYPE פעמיים של סכימת רגיסטר עם אפס וכתיבה לרגיסטר שני. בפעם הראשונה מבצעים לרגיסטר הראשון, ואז מעבירים את המידע אחורה ולא נותנים לPC להתקדם (כמו הפתרון שלנו) ומחליפים בין שני כתובות הרגיסטרים ומבצעים את האותה פקודה שוב ללא העיכוב הפעם.

דגשים – זיהוי הבעיות, סתירת פקודות קיימות, שליטה ע"י העברת נתונים נאותה (קדימה ואחורה), עמידה בהגדרות השאלה.

ג. (10 נקודות) כיצד תשתנה מערכות הFORWARDING השונים (לציין את שניהם!)?

ראשית נשים לב כי יחידת ה FORWARDING של שלב הEX חייבת להתעדכן שכן אם בשלב SWAP1 אנו מתעתדים לכתוב לתוך הרגיסטר השני הרי שכפי שהמערכת כרגע, יחידה זו תכניס את הערך החדש לALU בזמן SWAP2. ולכן צריך שיחידת הFORWARDING תעשה את עבודתה נאמנה בכל שאר הפקודות מלבד פקודת SWAP בחלק השני! בחלק זה היא תבטל את הבדיקה משלב הMEM בלבד!

כלומר קו הבקרה תבדוק אם מדובר בSWAP2 לא לאפשר FORWARD משלב הMEM. מכיוון שדבקנו במערכות קיימות **רוב מוחלט** המערכת נשארת ללא שינויים.

FORWARDING החדש בשלב הDECODE מיועד רק לפקודות BRANCH ואינו רלוונטי. הוא אינו מחליף ערכים רק בוחן השוואה. הוא יודע לבצע השוואה לפי יעדי כתיבה של פקודות שקדמו לו. מכיוון שהכנסנו את הפקודה שלנו לתוך פורמט פקודות קיימות לא יצרנו בעיות נוספות.

ד. (5 נקודות) כיצד תשתנה ה HAZARD DETECTION UNIT ? האם המערכת עדיין תצליח לפתור את כל בעיות וסוגי הHAZARDS שלמדתם בכיתה? במידה ויש השוואות נוספות שצריכים להעשות ציינו זאת. במידה ויש ייעול ציינו זאת.

DATA HAZARDS אינם קשורים למערכת זו וכבר נפתרו.

LOAD HAZARDS נפתרים בצורה הרגילה (BUBBLE INSERTION) אך ניתנים לייעול שכן רק רגיסטר אחד באמת צריך להיבדק בשלב של SWAP1 שכן הרגיסטר השני ייקרא שוב בSWAP2 כאשר הערך כבר יעודכן.

אין שום שינוי למערכת מלבד הערה זו.

ה. (5 נקודות) מה המחיר בזמן ההרצה של תוכנית כלשהי אם 5% מסך הפקודות בתוכנית (התוכנית מבצעת הרבה מאוד פעולות של SORT) הם מסוג זה? האם המחיר הנ"ל תלוי בHAZARDS חדשים מתוקף הפקודה החדשה? כיצד?

אין שום תלות בHAZARDS חדשים. נוסיף מחזור יחיד בלבד לפקודות הSWAP על פני שאר הפקודות ולכן אם 5% מהזמן יש פקודה מסוג זה הרי שזמן ההרצה מתארך כך ש

$T_{new} = T_{old} * 95\% + T_{old} * 5\% * 2$ תחת הזנחת מילוי הצינור.

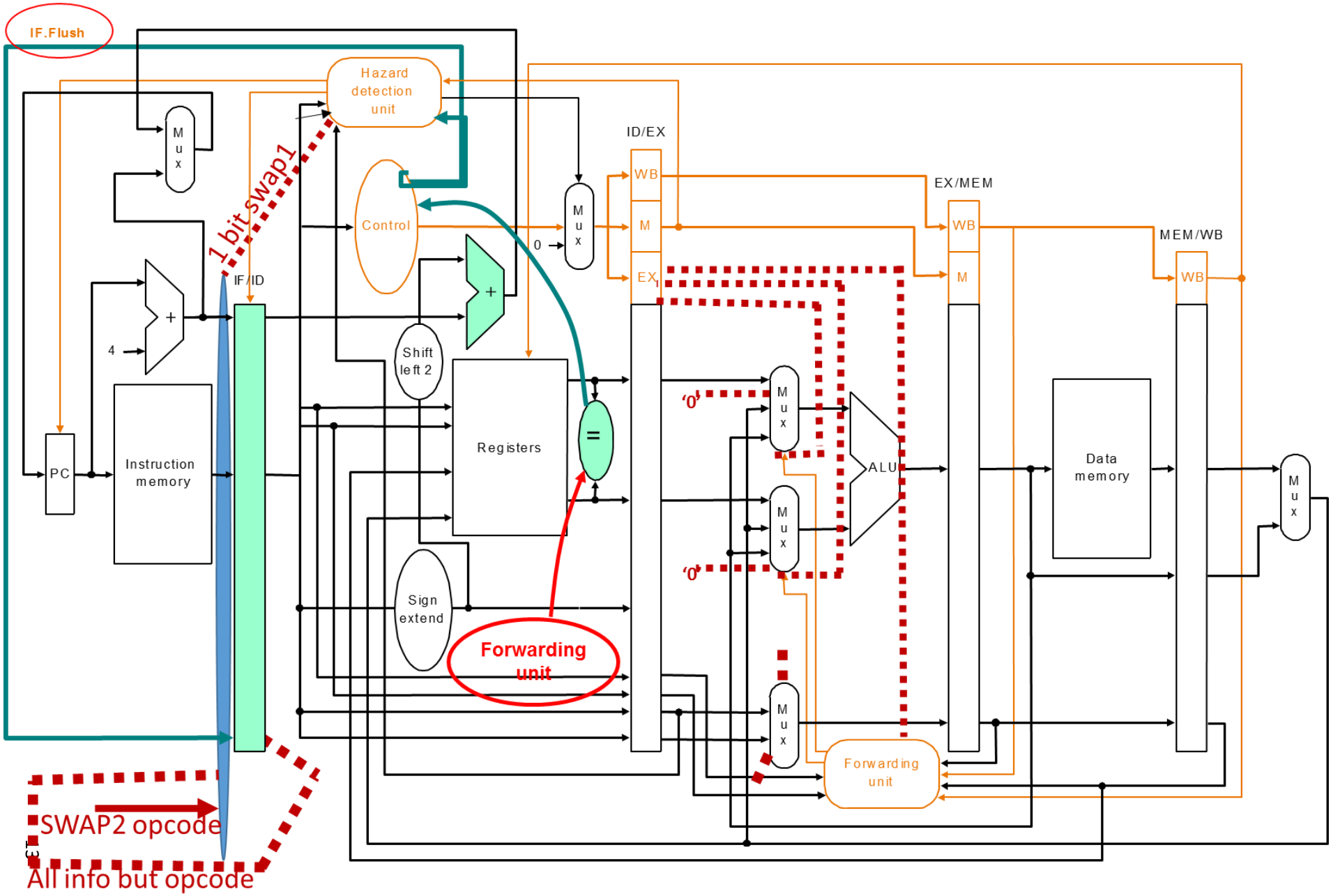
ו. (5 נקודות) האם ניתן לממש את הפקודה הנ"ל בין שני ערכים האגורים בזכרון ולא בקובץ הרגיסטרים? כיצד? ענו תחת הגבלות מעין אלו לעיל (ללא שינויים במערכים קיימים למעט קוי בקרה ובוררים).

ציינו מהם הבעיות העקרוניות וכיצד חייבים לפתור אותם. בכל מקרה אין לחרוג ממחיר של 2 מחזורים לפקודת הSWAP .

ללא שינויים לא ניתן לעמוד בתנאים. לא ניתן לבצע קריאה, כתיבה, קריאה , כתיבה בשום אופן מהזכרון ב2 מחזורים. ולכן גם לא ניתן לקרוא ולכתוב לשני מקומות אחרים בזכרון, קל וחומר להחלפה בין שניים.

ז. (5 נקודות) האם ניתן לבצע זאת ע"י שינויים של מערכת הזכרון? למרות השינויים האם ניתן לבצע זאת במחזור יחיד? באיזה מידה זה שונה מאשר ביצוע שינויים מעין אלו לצורך ביצוע הפקודה בקובץ הרגיסטרים (רמז, תזמון השעון).

עם שינויים הכל כבר אפשרי. הוספת פורט קריאה נוסף, העברה של יותר מיעד קריאה אחד וכן כתיבה מתוזמנת. הבעיה היא הכתיבה שאיננה מתוזמנת שעון. נצטרך לוודא שתזמון הקריאה והכתיבה נעשה בצורה שווה במקביל ושלא יתבצע SKEW בין שני חלקי הפקודה באותו מחזור שעון כך שייווצר מרוץ. צריך ליצור חוצץ בין מערכת הכתיבה לזכרון לבין מערכת הקריאה שיתוזמן כך שלא ייווצר מצב בשום פקודה של קריאה וכתיבה מהזכרון באותו זמן כך שמידע שנקרא במחזור אחד לא יכתב למקום אחר במחזור הבא (יצירת LOOP ללא חוצץ הוא בעייתי).


IF.Flush
Hazard detection unit
1 bit swap1
Control
ID/EX
EX/MEM
MEM/WB
WB
M
EX
Mux
0
IF/ID
4
Shift left 2
PC
Instruction memory
Registers
=
'0'
ALU
Data memory
Sign extend
Forwarding unit
SWAP2 opcode
All info but opcode

<u>שאלה ג (30 נקודות)</u>

שאלת תכנון ומימוש בשפת תכן Verilog:

תיאור כללי של הבעיה:

- נדרש: רכיב המממש מיון-בועות "bubble sort" הממומש לפי האלגוריתם שנידון בהרצאה.
- הרכיב מקבל מטריצה דו מימדית בגודל 8 על 32 בשם in (כלומר שמונה מלים בנות 32 ביט כל אחת). הרכיב ממיין את המלים לפי הסדר ומוציא מטריצה זהה בגודלה ומסודרת out.
- חובה להשתמש במבנה generate ולהשתמש בהפניות היררכיות בקוד.
- חובה להגדיר את מספר המלים וגודלן כפרמטר.

סעיפים:

1. (25 נק) תכנן את המעגל צירופית לחלוטין.
2. (5 נק) תאר בנקודות אילו שינויים היית עושה בכדי להפוך את הקוד לסינכרוני (עם שעון clk) ובמבנה pipeline. אין צורך לממש אך מספיק לרשום בנקודות עיקריות מה השינויים המרכזיים הדרושים.

<u>דגש: ניקוד משמעותי ירד לכתב לא ברור\ לא מסודר\ מקושקש מדיי - תעבדו עם עפרון או במחברת טיוטא ורק קוד סופי למסגר – רק מה שימוסגר ייבדק.</u>

<u>פתרון-</u> <u>זהו מקום לקוד סופי (טיוטא במחברת):</u>

מודול_ממיין_בועות:

פתרון:

```
module comp #(parameter NUM_ARG = 8, WIDTH = 32) (A, B, H, L);
  input wire [WIDTH-1:0] A,B;
  output wire [WIDTH-1:0] H,L;
//
  assign {H,L} = (A>B)?{A,B}:{B,A};
endmodule


module BubbleSort_COMB #(parameter NUM_ARG = 8, WIDTH = 32) (in, out);
input [WIDTH-1:0] in [NUM_ARG-1:0];
output  [WIDTH-1:0] out [NUM_ARG-1:0];

genvar i,j;

generate

 // start at the lowest level in the tree
 for (i=0; i < NUM_ARG-1 ; i=i+1) begin:gen_levels
   // do nodes at each level
   for (j=0; j < (NUM_ARG-i-1); j=j+1) begin:gen_nodes
     wire [WIDTH-1:0] out_low;
     wire [WIDTH-1:0] out_high;

     if (i==0 & j==0) begin// at the lowest level, use the module's inputs
         comp (in[0][WIDTH-1:0], in[1][WIDTH-1:0], out_high, out_low);
     end
     else  if (i==0) begin
         comp (in[j+1][WIDTH-1:0], gen_levels[0].gen_nodes[j-1].out_high, out_high, out_low);
     end
       // otherwise, use the outputs of the lower level
     else  if (j==0) begin
         comp (gen_levels[i-1].gen_nodes[0].out_low, gen_levels[i-1].gen_nodes[1].out_low, out_high, out_low);
     end
     else
         comp (gen_levels[i].gen_nodes[j-1].out_high, gen_levels[i-1].gen_nodes[j+1].out_low, out_high, out_low);
     end
   end
 //end

assign  out[7][WIDTH-1:0] = gen_levels[1].gen_nodes[6].out_high;
assign  out[6][WIDTH-1:0] = gen_levels[2].gen_nodes[5].out_high;
assign  out[5][WIDTH-1:0] = gen_levels[3].gen_nodes[4].out_high;
assign  out[4][WIDTH-1:0] = gen_levels[4].gen_nodes[3].out_high;
assign  out[3][WIDTH-1:0] = gen_levels[5].gen_nodes[2].out_high;
assign  out[2][WIDTH-1:0] = gen_levels[6].gen_nodes[1].out_high;

assign  out[1][WIDTH-1:0] = gen_levels[7].gen_nodes[0].out_high;
assign  out[0][WIDTH-1:0] = gen_levels[7].gen_nodes[0].out_low;

endgenerate

endmodule
```

סעיף ב:

```
reg [WIDTH-1:0] Out_high;
reg [WIDTH-1:0] Out_low;

if (i==0 & j==0) begin// at the lowest level, use the module's inputs
    comp (in[0][WIDTH-1:0], in[1][WIDTH-1:0], out_high, out_low);
    always @(posedge clk) Out_low<=out_low;
end
else  if (i==0 & j==7) begin
    comp (in[j+1][WIDTH-1:0], gen_levels[0].gen_nodes[j-1].out_high, out_high, out_low);
    always @(posedge clk) Out_low<=out_low;
    always @(posedge clk) Out_high<=out_high;
end
    else  if (i==0) begin
    comp (in[j+1][WIDTH-1:0], gen_levels[0].gen_nodes[j-1].out_high, out_high, out_low);
    always @(posedge clk) Out_low<=out_low;
end
  // otherwise, use the outputs of the lower level
else  if (j==0 & i != 7) begin
    comp (gen_levels[i-1].gen_nodes[0].out_low, gen_levels[i-1].gen_nodes[1].out_low, out_high, out_low);
    always @(posedge clk) Out_low<=out_low;
end
else  if (j==0 & i==7) begin
    comp (gen_levels[i-1].gen_nodes[0].out_low, gen_levels[i-1].gen_nodes[1].out_low, out_high, out_low);
    always @(posedge clk) Out_low<=out_low;
    always @(posedge clk) Out_high<=out_high;
end
else if (j>NUM_ARG-i-1) //Main changes in the code here for SEQ
    always @(posedge clk) Out_high<=gen_levels[i-1].gen_nodes[j].Out_high;
else
    comp (gen_levels[i].gen_nodes[j-1].out_high, gen_levels[i-1].gen_nodes[j+1].out_low, out_high, out_low);
    always @(posedge clk) Out_low<=out_low;
end
```