אוניברסיטת בר-אילן

הפקולטה להנדסה

BAR-ILAN UNIVERSITY

Engineering Faculty

תכן לוגי ומבוא למחשבים

תשע"ז מועד א

83-253

- מרצה : פרופ' שמואל וימר
- מתרגל : מר יוסף לונדון ,מר איתמר לוי ,גב ספיר ארליך
- **חומר עזר מותר:** שקפי ההרצאות כולל הערות והסברים שנרשמו במהלך ההרצאות וספר הקורס **<u>בלבד</u>**. כל חומר אחר, ובכלל זה חומר תרגול, **<u>אסורים בתכלית</u>**!
- משך המבחן: **<u>שלש שעות</u>**
- 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> (40 נקודות)

נתון מעבד MIPS בעל מעבר נתונים ובקר המתואר בתרשימים 1 ו 2.

נדרש להוסיף תמיכה בפקודה חדשה **swap $rs, $rt** שתפקידה להחליף את התוכן של שני אוגרים כדלקמן:

RegFile[**rs**] ← RegFile[**rt**] ; RegFile[**rt**] ← RegFile[**rs**].

פורמט הפקודה הינו מסוג R-type, כאשר פורמט הפקודה מחייב ש **rs = rd**. <u>מימוש הפקודה בחומרה מנצל עובדה זאת.</u>

(כשנאמר שפורמט הפקודה מסוג R-type, הכוונה לכך שיש rs,rd,rt באותם ביטים כמו בR-type. ניתנה האפשרות לתת לפקודה opcode חדש, או (יותר מסובך אך אפשרי) לקבוע אותה כ R-type עם opcode=0, ובמקרה זה צריך גם לקבוע את שדה הfunct, ולתת לבקר אפשרות לדעת שאכן זו פקודת swap במקרה שהוספה התפצלות מצבים חדשה)

1. (10 נקודות) במדה ונדרש, הוסיפו ו/או שנו בתרשים 1 את כל החומרה הנדרשת למימוש הפקודה, הן את המעגלים והן את האותות (נתונים ובקרה). **<u>אסור לשנות את ה Register File (RegFile) בשום אופן!</u>** יש להסביר במחברת באופן מילולי ומפורט את כל השינויים שנעשו על גבי הסכמה. במדה ואין שינויים, יש להסביר כיצד מבצעת החומרה את הפקודה בכל זאת.
2. (10 נקודות) שנו את מכונת המצבים של הבקר בתרשים 2 כך שהפקודה החדשה תתבצע במספר מינימלי של מחזורי שעון. במדה ואתם מוסיפים או משנים מצבים יש לרשום בתוכם את ערכי אותות הבקרה וה enables הרלוונטיים. כמה מחזורי שעון לוקח בצוע הפקודה?

תשובה: משתמשים בשני מחזורי שעון WB1 ו WB2 לבצוע. במחזור WB1 נשתמש בערך $rt, שמתייצב באוגר B, לצורך כתיבתו ב rd (ועלכן גם ב rs כמתחייב מדרישת הפורמט של הפקודה). שימו לב שהכתיבה ל RegFile תתבצע בפועל רק בעת הגעת פולס השעון. ניתוב הערך של B נעשה ע"י הרחבת ה MUX המתאים (כניסה 3). בינתיים האוגר A מכיל את $rs. במחזור השעון הבא יש להעביר את ערכו של A ל rt. הדבר נעשה ע"י קביעת כתובת הכתיבה ל RegFile כ rt וניתוב A דרך כניסה 2 של ה MUX המתאים. נדרשים 4 מחזורי השעון לפקודת swap.

אפשרות נוספת, הינה לבצע במקביל ל WB1 חיבור ALU result= A+0 כלומר גם פעולת EXE. זה מצריך כניסת אפס ל MUX B ועקב כך הרחבת ALUSrcB לשלש סיביות.

3. (3 נקודות) האם ניתן לממש פקודה כזאת בדיוק ב single cycle MIPS, ובהתאם לאותן דרישות על איסור השינוי ב RegFile ? נמקו בפירוט מלא את תשובתכם. תשובה ללא נימוק לא תתקבל.

תשובה: הדבר בלתי אפשרי משום שפתרון הבעיה ללא הגדלת מספר כניסות הכתיבה ל RegFile מחייב אגירת ביניים של ערכים ו single cycle MIPS איננו מכיל כאלה. מאידך, הוספת אוגר לאכסון ערך ביניים מחייבת לפחות שני מחזורי שעון לפקודה, בלתי אפשרי ב single cycle MIPS.

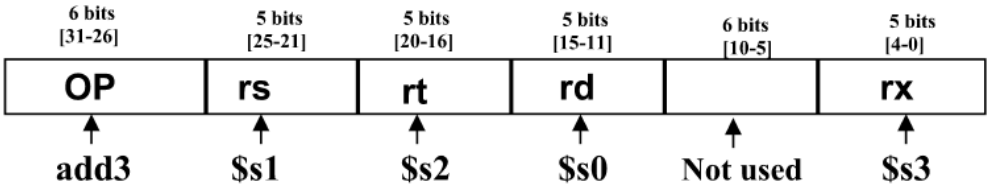
4. (2 נקודות) נתון קוד assembly של אלגוריתם מסוג bubble למיון מערך מספרים שלמים בזיכרון. באיזה אופן פקודת ה swap מייעלת את בצוע התכנית?

תשובה: ללא פקודת ה swap נדרשת החלפת ערכים בזיכרון תוך חישוב כתובות זיכרון עבור כל החלפה. שימוש ב swap מאפשר החלפת ערכים בין האוגרים וכתיבתם לשתי מילים עוקבות בזיכרון מבלי לחשב כתובות.

<u>קיבלנו גם</u>: עבור מימוש פקודת הswap כפסאודו-פקודה היה צריך להשתמש ב3 פקודות אסמבלי עבור ההחלפה

בין ערכי הרגיסטרים (תוך שימוש ברגיסטר ביניים), אך בעזרת הוספת swap למעבד ניתן לבצע את ההחלפה בערכים בפקודת אסמבלי יחידה, ולא נדרש רגיסטר ביניים פנוי.

כעת נדרש להוסיף פקודה המסכמת שלשה מספרים **add3 $s0, $s1, $s2, $s3**, כאשר השדות בפקודה הינם כדלקמן:



5. (8 נקודות) במדה ונדרש, הוסיפו ו/או שנו בתרשים 3 את כל החומרה הנדרשת למימוש הפקודה, הן את המעגלים והן את האותות (נתונים ובקרה). **<u>אסור לשנות את ה ALU בשום אופן!</u>** יש להסביר במחברת באופן מילולי ומפורט את כל השינויים שנעשו על גבי הסכמה. במדה ואין שינויים, יש להסביר כיצד מבצעת החומרה את הפקודה בכל זאת.
6. (7 נקודות) שנו את מכונת המצבים של הבקר בתרשים 4 כך שהפקודה החדשה תתבצע במספר מינימלי של מחזורי שעון. במדה ואתם מוסיפים או משנים מצבים יש לרשום בתוכם את ערכי אותות הבקרה וה enables הרלוונטיים. כמה מחזורי שעון לוקח בצוע הפקודה?

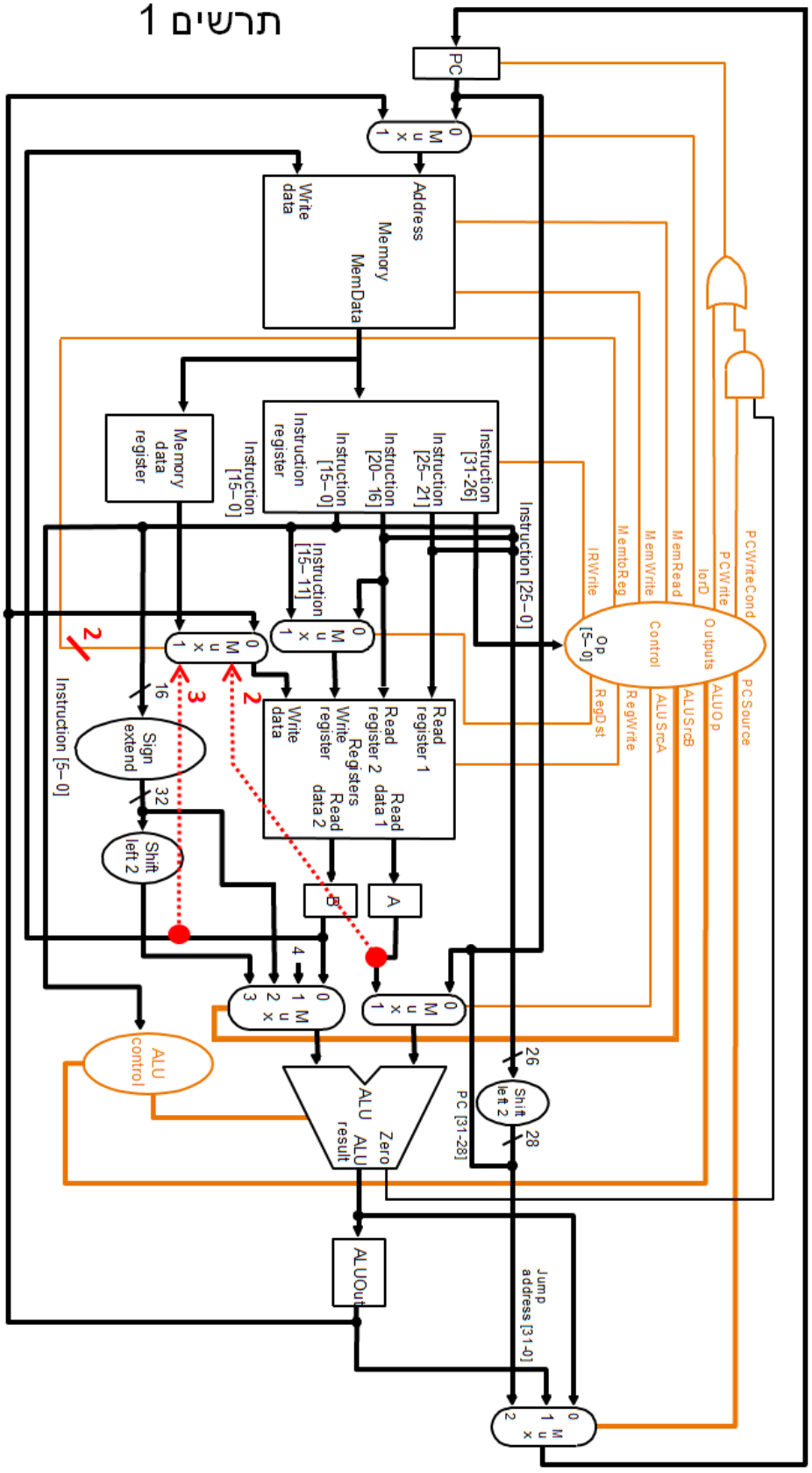
תשובה: הרעיון הינו להשתמש בשני מחזורי שעון EX1 ו EX2 לבצוע. מחזור EX1 דואג לסכם את $rs ($s1) עם $rt ($s2), כלומר ALUout ← A+B. התוצאה הזאת תשמש כאופרנד הראשון למחזור EX2 בכניסה A, דבר המחייב את הרחבת ה MUX בכניסת A של ה ALU, והרחבת אות הבקרה ALUSrcA לשתי סיביות. יש כמובן לעדכן את ערך האות מסיבית אחת לשתיים בכל המצבים הקיימים.

את המחובר השלישי $rx ($s3) המשמש כאופרנד השני לחבור הבא יש להכין ע"י העברתו לכניסה B, דבר המחייב ראשית כתיבתו לתוך האוגר B. לבסוף, במחזור EX1 rt נכתב לתוך B, ואילו rx במחזור EX2. זה דורש ניתוב ע"י MUX מתאים בכניסת Read register 2 של ה RegFile ואות בקרה מתאים שנקרא לו ReadSrc. יש לשים לב שהבקרה של כל הפקודות הנדרשות לערך של rt נדרשת כעת להחליט בינו לבין rx ע"י קביעת ערכו של ReadSrc 0/1. זה מחייב עדכון במצבים הקיימים של הפקודות השונות. נדרשים 5 מחזורי השעון לפקודת add3.

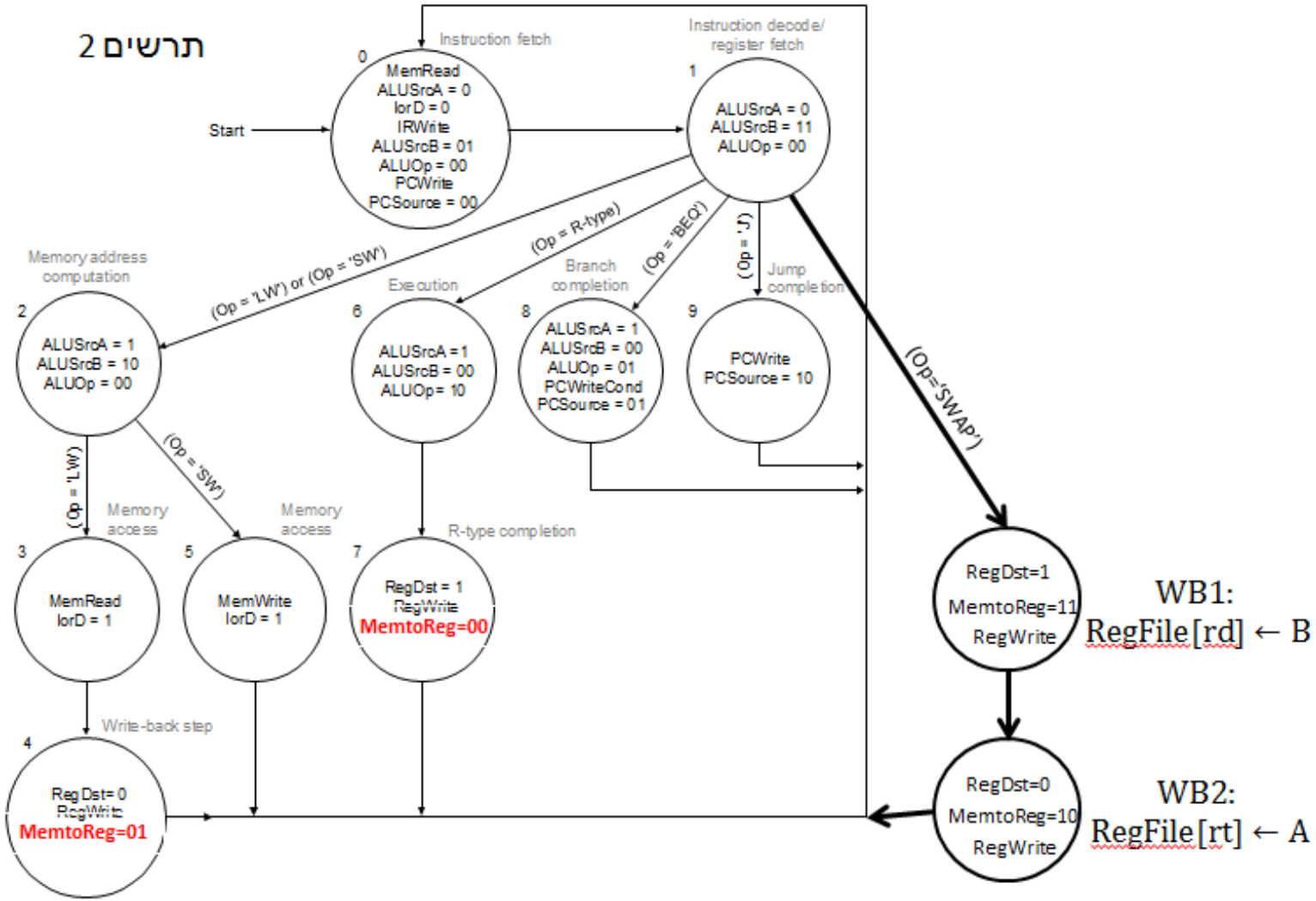
<u>שימו לב</u>: הדרישה היא לא לשנות את ה*ALU*, וזה אומר שגם אסור להוסיף יחידות אריתמטיות חדשות (*ALU* נוסף, *Adder* וכד'). (הורדתי על זה 9 נקודות בסה"כ (6 בסעיף 5, ו3 בסעיף 6, [אם מימוש הבקר היה נכון בהתחשב בחומרה שציירו בסעיף 5, אחרת ירדו יותר נקודות בסעיף 6])

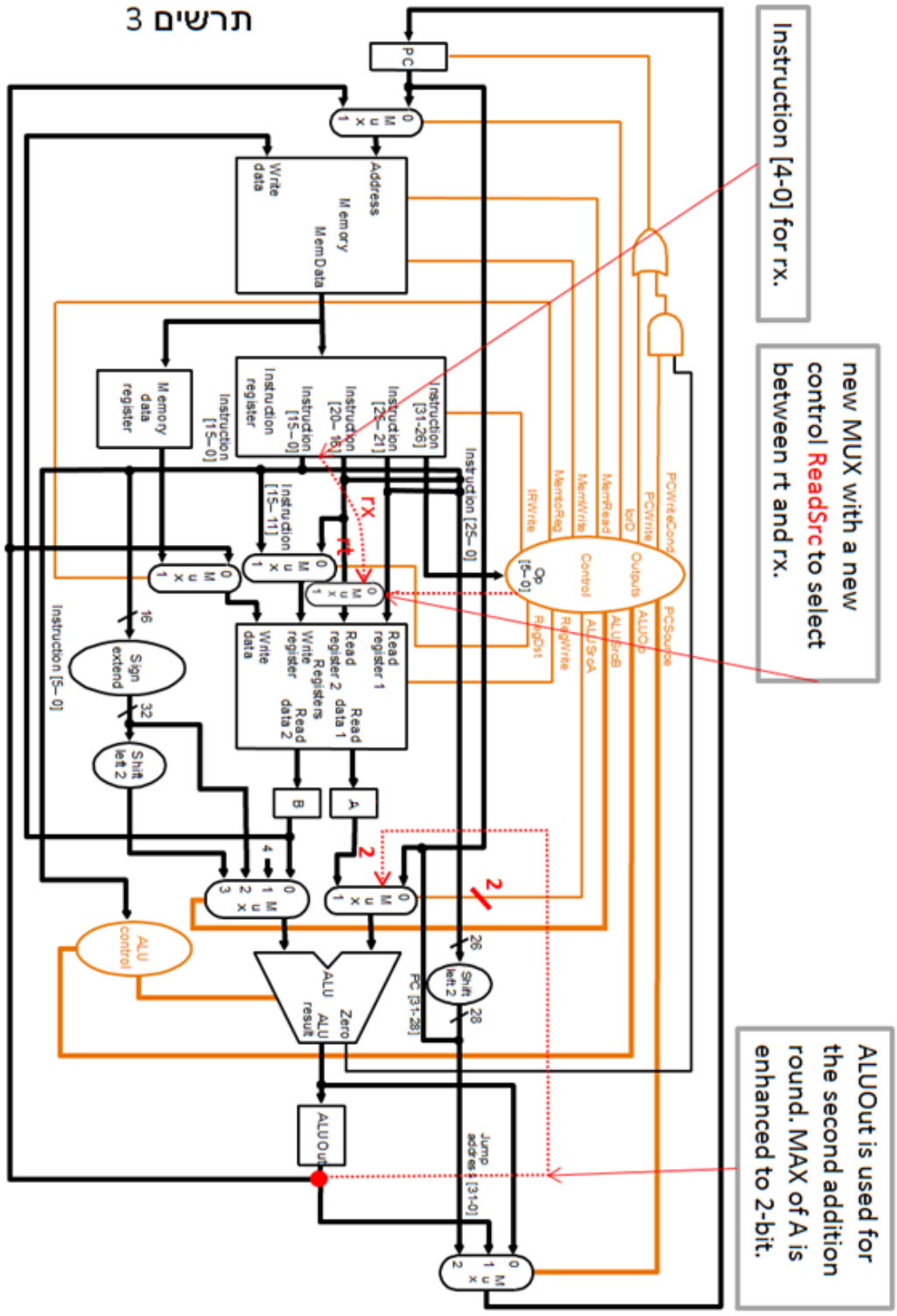
תרשים 1

The outputs of A and B should be connected to the MUX controlled by MemtoReg, which must be 2 bits.

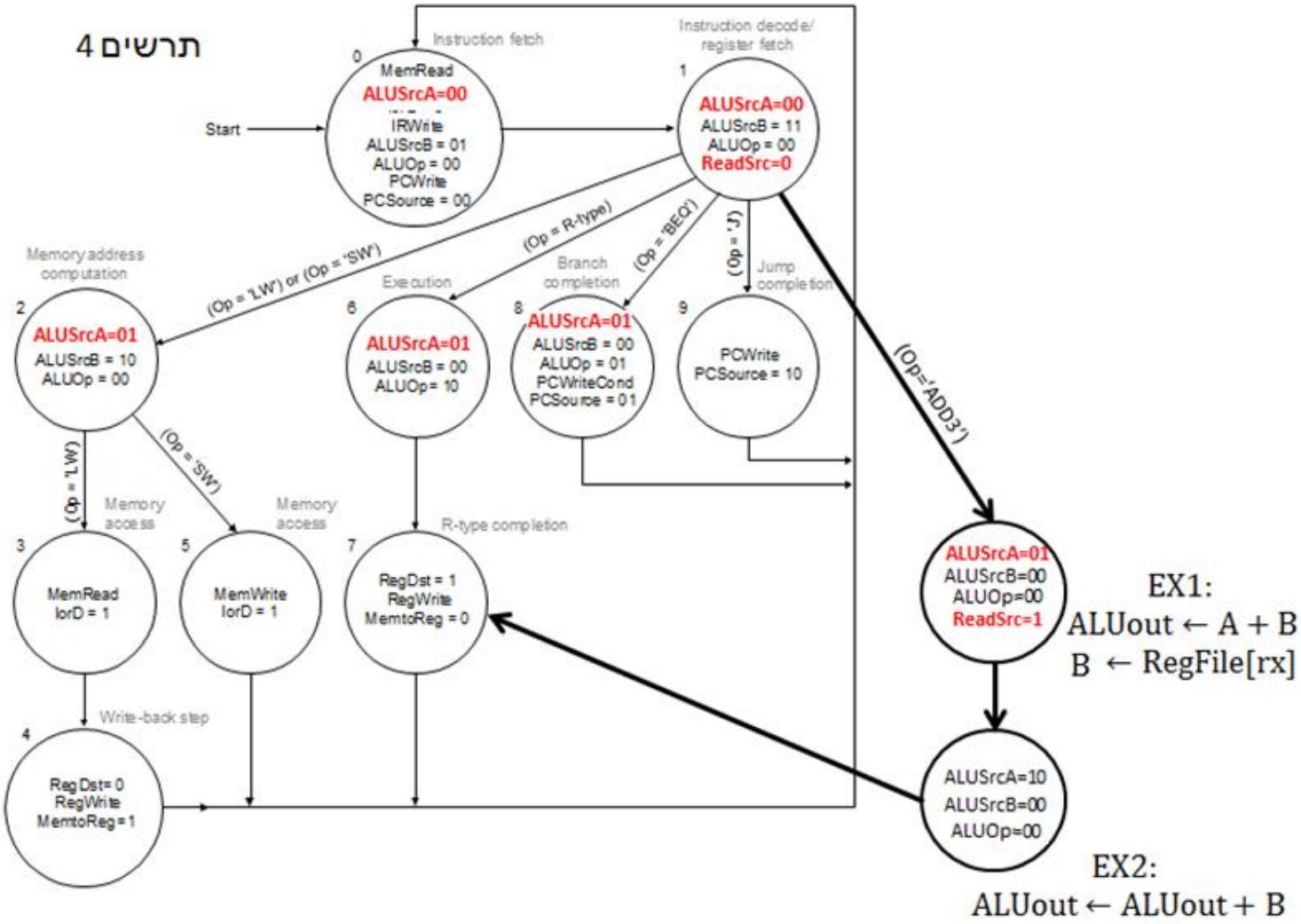


תרשים 2


Start
0
Instruction fetch
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
1
Instruction decode/
register fetch
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
(Op='SWAP')
2
Memory address
computation
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
Execution
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
Branch
completion
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
Jump
completion
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
3
Memory
access
MemRead
IorD = 1
5
Memory
access
MemWrite
IorD = 1
7
R-type completion
RegDst = 1
RegWrite
MemtoReg=00
4
Write-back step
RegDst= 0
RegWrite
MemtoReg=01
RegDst=1
MemtoReg=11
RegWrite
WB1:
RegFile[rd] ← B
RegDst=0
MemtoReg=10
RegWrite
WB2:
RegFile[rt] ← A


תרשים 3
Instruction [4-0] for rx.
new MUX with a new control ReadSrc to select between rt and rx.
ALUOut is used for the second addition round. MAX of A is enhanced to 2-bit.
PC
Address
Memory
MemData
Write data
Memory data register
Instruction register
Instruction [31-26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–0]
Instruction [15–11]
Instruction [25–0]
Instruction [5–0]
rx
rt
PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Op [5–0]
Outputs
Control
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
PCSource
Sign extend
16
32
Shift left 2
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
A
B
4
2
ALU control
ALU
Zero
ALU result
ALUOut
26
28
PC [31-28]
Jump address [31-0]
M u x

תרשים 4


Start
0
MemRead
ALUSrcA=00
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Instruction fetch
Instruction decode/
register fetch
1
ALUSrcA=00
ALUSrcB = 11
ALUOp = 00
ReadSrc=0
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
(Op='ADD3')
Memory address
computation
2
ALUSrcA=01
ALUSrcB = 10
ALUOp = 00
Execution
6
ALUSrcA=01
ALUSrcB = 00
ALUOp = 10
Branch
completion
8
ALUSrcA=01
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
Jump
completion
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Write-back step
4
RegDst = 0
RegWrite
MemtoReg = 1
ALUSrcA=01
ALUSrcB=00
ALUOp=00
ReadSrc=1
ALUSrcA=10
ALUSrcB=00
ALUOp=00
EX1:
ALUout ← A + B
B ← RegFile[rx]
EX2:
ALUout ← ALUout + B

<u>שאלה ב'</u> (35 נקודות)

נתונה התכנית שלהלן המיועדת ל MIPS ומשתמשת ב delay slots.

```
loop:
      LW    R4,  0(R3) // 1
     ADDI   R5,  R4,  0 // 2
      SLR   R5,  R5,  1 // 3
      SLL   R5,  R5,  1 // 4
b1:
      BEQ   R4,  R5,  b2 // 5
     ADDI   R3,  R3,  4 // 6
     ADDI   R2,  R2,  1 // 7
b2:
      BNEZ  R1,  loop // 8
      SUBI  R1,  R1,  1 // 9
```

הערה:
SLL - Shift Left Logical
SLR - Shift Right Logical
פקודות ה- shift אינן ציקליות

נתונים ערכי ההתחלה הבאים: $R1 = n,\ R2 = 0,\ R3 = p$ (R3 מצביע לראש מערך של מספרים שלמים).

1. (5 נקודות) מה בדיוק מבצעת התכנית הנ"ל? מהו ערכו של R2 בעת היציאה מהלולאה?

תשובה: התוכנית מקבלת מערך בכתובת p וסופרת כמה מבין n האיברים הראשונים בו הינם אי זוגיים. בסיום התוכנית R2 מכיל את כמות האיברים האי זוגיים. בדיקת הזוגיות נעשית ע"י הזזה ימינה ואחר כך שמאלה. במידה ו LSB=0 במקור, ערכו של המספר שהיה זוגי מלכתחילה איננו משתנה. לו LSB=1 במקור, המספר שהיה ב מלכתחילה אי זוגי ייהפך לזוגי.

התכנית הנ"ל מתבצעת במעבד MIPS כמתואר בתרשים 5. נתון שלמעבד יש גם יחידות קידום נתונים (forwarding) ו hazard detection שאינן נראות בתרשים.

2. (5 נקודות) האם התכנית תתבצע באופן תקין? נמקו את תשובתכם באופן מפורט. במדה והתשובה שלילית ציינו במדויק אילו תיקונים נדרשים בתכנית ובאיזה מקומות.

תשובה: התוכנית תבצע באופן שגוי.
הסיבה לכך הינה שלפי התרשים הבדיקה לקיום קפיצה מותנית מתבצעת בשלב ה MEM של הצינור, ובצינור כבר נמצאות שלש פקודות נוספות, גם לו הקפיצה אמורה הייתה להתבצע (taken). בכדי שהתוכנית תתבצע נכון יש לדאוג ש:

1. הפקודה 7 תתבצע רק במדה והקפיצה איננה מתרחשת. הדבר יתאפשר ע"י הוספת שתי פקודות NOP לפני הפקודה 7.
2. שתי פקודות בזיכרון הרשומות אחרי פקודה 9 תקראנה ותתחלנה להתבצע למרות שאינן שייכות כלל לתכנית! בכדי למנוע זאת יש להוסיף שתי פקודות NOP אחרי פקודה 9.

3. (10 נקודות) איזה שינויי חומרה יש לבצע על מנת שהתכנית המקורית תתבצע בכל זאת מבלי לשנות בה דבר? הסבירו באופן מפורט כל שינו שאתם מבצעים, ציירו אותו על גבי תרשים 5, ונמקו כיצד השינוי פותר את הבעיה.

תשובה: מאחר וכל אחת משתי הקפיצות המותנות הרשומות בתכנית ישנו delay slot יחיד, יש לדאוג שהפקודה הזאת תתבצע, אבל שום דבר נוסף לא יכנס לצינור לפני קבלת ההחלטה על קפיצה. הדבר יתאפשר ע"י התרשים החדש. השינויים הנדרשים הינם:

1. הקדמת בדיקת תנאי הקפיצה משלב ה MEM לשלב ה ID.
2. הקדמת חישוב כתובת הקפיצה משלב ה EXE לשלב ה ID.

כמובן ששני השינויים הנ"ל מחייבים שימוש במנגנון ה forwarding.

יש לשים לב לא ליצור בטעות מנגנון של flash ב IF למקרה שהקפיצה taken! הדבר יוביל להשמדה מוטעית של הפקודה המצויה ב delay slot, פקודה מועילה שאמורה להתבצע תמיד.

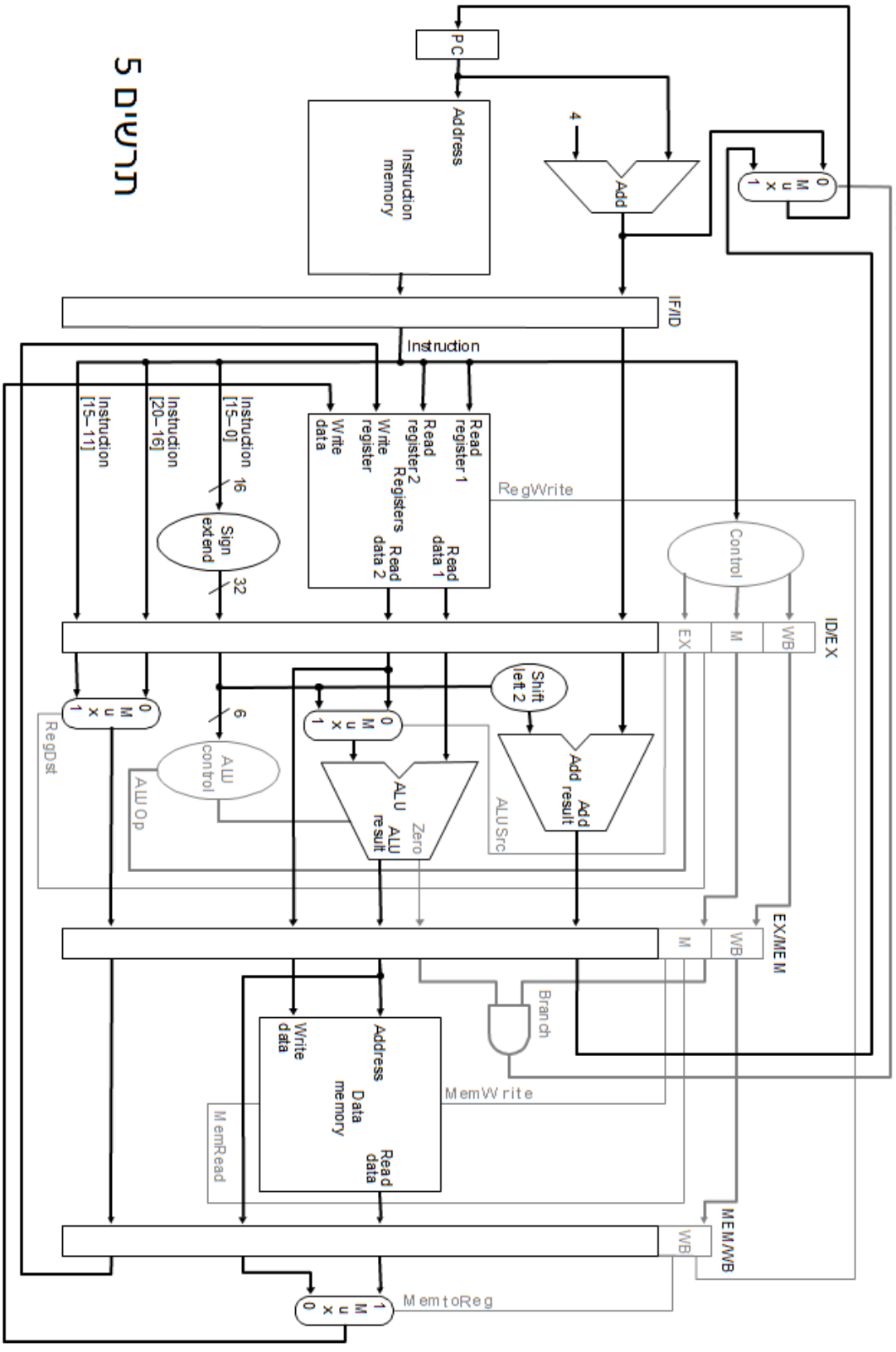
4. (10 נקודות) נניח שהאיבר הראשון במערך המספרים הוא 6 והשני 11. רשמו בטבלה שלפניכם את כל הפקודות המבוצעות במהלך ממחזורי השעון ככל שהטבלה מאפשרת למלא. לשם נוחות הפקודות בקוד התכנית ממוספרות ועמודה INSTR ניתן לרשום את המספר של הפקודה.

תשובה: יש לשים לב למצב של load hazard שמתרחש בפקודה 2 וגורם להשהיית מחזור שעון בפקודות 2 ו 3. בהמשך, משום שתנאי הקפיצה מתקיים (6 מספר זוגי), פקודה 7 איננה מתבצעת. בלולאה השנייה פקודה 7 כן מתבצעת ומונה האי זוגיים מתעדכן. כמו כן פקודות הקפיצה המותנית נמצאות בצינור רק שני מחזורי שעון, שאחריהן הן מסתיימות.

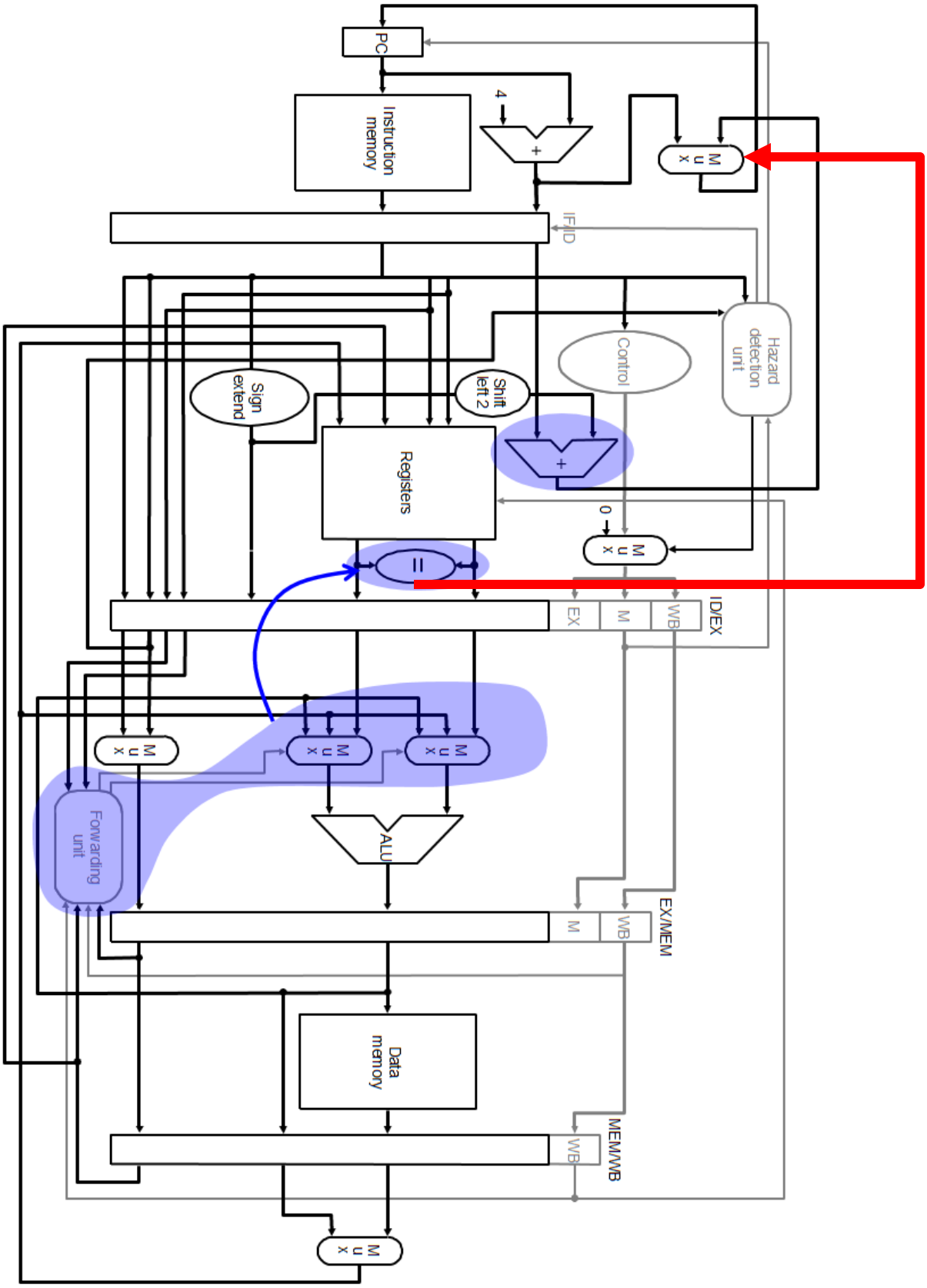
CLK	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
INSTR																					
1	IF	ID	X	M	W																
2		IF	ID	ID	X	M	W														
3			IF	IF	ID	X	M	W													
4					IF	ID	X	M	W												
5						IF	ID														
6							IF	ID	X	M	W										
8								IF	ID												
9									IF	ID	X	M	W								
1										IF	ID	X	M	W							
2											IF	ID	ID	X	M	W					
3												IF	IF	ID	X	M	W				
4														IF	ID	X	M	W			
5															IF	ID					
6																IF	ID	X	M	W	
7																	IF	ID	X	M	W

5. (5 נקודות) נתון מערך של מאה מספרים טבעיים רצופים. כמה מחזורי שעון תארך התכנית? הסבירו במדויק את חישוביכם.

תשובה: במחזור לולאה של מספר אי זוגי מתבצעות כל הפקודות, ובנוסף שינו שיהוי של מחזור אחד בגלל load hazard, סה"כ 10 מחזורי שעון. במחזור של מספר זוגי הפקודה 7 איננה מתבצעת. על כן בסה"כ $(10 + 9) \times 50 = 950$ מחזורי שעון. נדרשים עוד 5 מחזורי שעון בשל הפקודה הראשונה שממלאת את הצינור, ובסה"כ 955 מחזורי שעון.


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

תרשים 5


PC
Instruction memory
4
+
M u x
IF/ID
Hazard detection unit
Control
Sign extend
Shift left 2
Registers
+
0
M u x
=
EX
M
WB
ID/EX
M u x
M u x
M u x
Forwarding unit
ALU
M
WB
EX/MEM
Data memory
WB
MEM/WB
M u x

3) שאלת תכנון ומימוש בשפת תכן Verilog:

תיאור כללי של הבעיה:

- נדרש: מעגל לוגי המקבל טורית סדרה IN, בין כל תת סדרה באורך n או m קיים "חוצץ" (שמוגדר להיות כניסת לכל הפחות אפס אחד).

$$\ldots,\ 0,\ 0,\ n,\ x_1,\ x_2,\ \ldots,\ x_n, 0, 0, 0, 0,\ m,\ y_1,\ y_2,\ \ldots, y_m,\ 0,\ 0,\ \ldots$$

$$OUT = \ldots\ldots\ldots\ldots\ldots, \sum_n x_i, \ldots\ldots\ldots\ldots\ldots, \sum_m y_i, \ \ldots\ldots\ldots\ldots$$

- ומחשב את הסכומים : $\text{Ready} = 0,0\ldots\ldots,0,\ \ 1\ \ ,0,0\ldots\ldots,0,\ \ 1\ \ ,0,0,\ldots\ldots 0$
- כלומר המוצא OUT מוציא סכום (ובין לבין יכול להוציא "זבל") וקיים ביט בקרה נוסף שמציין שתוצאת החישוב מוכנה Ready=1.
- לשימושכם רק רגיסטר בגודל N_ADD ביטים (תניחו חזקה של 2).
- אלמנטים מהסדרה (כמו n ו- m) ובפרט x'ים ו- y'ים בגודל N_ARGS ביטים. תניחו שכל האלמנטים חיוביים. הקוד מודולרי ומוגדר לפי הפרמטרים N_ADD ו- N_ARGS. הגודל של המשנים יכול ליצור מצבים "לא אפשריים" לחישוב. מצאו את התנאי שמביא למצבים אלו (תלוי גם ב גודל הסדרה- m או n). כאשר מגיעה סדרה לא חוקית (שנוצרת ממצב "לא אפשרי") אתם מוציאים סיגנל Valid =0 וממתינים לתום הסדרה (מצב HOLD).
- בין כל סדרה לסדרה יש להמתין ל"חוצץ" (שמוגדר להיות כניסת **<u>לכל הפחות</u>** אפס אחד, מצב INVALID). רק אז מתחילים בפעולה שוב בכניסת m (או n).
- שלבו ריסט Reset אסינכרוני במערכת.
- ניתן להניח שהכניסה IN מסונכרנת לשעון (clk).

סעיפים:

1. (5 נק) תכנון מסלול נתונים בעזרת רכיבים סטנדרטיים באופן התנהגותי (behavioral) או מבני (Structural) המתייחסים ליחידות מידע בסיסיות של אלמנט מתוך מילה. חיבורים ופעולות אריתמטיות צריכים להיות מסונכרנים לשעון.
2. (ללא ניקוד אך דרוש) הגדרת קווי בקרה וסטטוס – דרישות של מסלול הנתונים מהבקר
3. (15 נק) תכנון בקר בשיטת FSM - תארו מכונת מצבים **<u>מלאה</u>** (MEALY או MOORE כרצונכם) וממשק עם סיגנלי הבקרה. כמובן שעליכם להוסיף סיגנלים ומצבים.

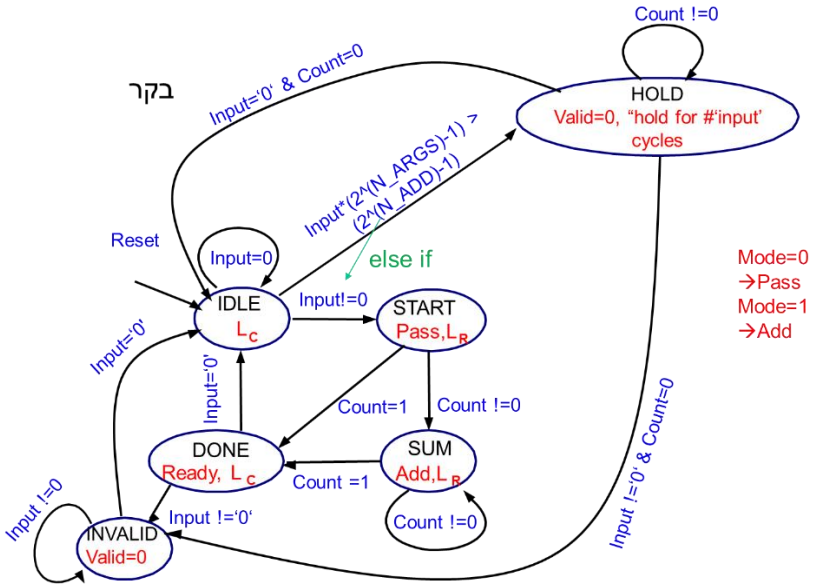
4. (5 נק) בנו TOP LEVEL שמכיל אינסטנסיאציה של הבקר ושל נתיב הנתונים (data-path). שימו לב שהאינסטנסיאציה עם פרמטרים ל BITS_N, ADD_N (אתם יכולים לתת ערך דיפולטיבי כרצונכם).

5. (5 נק) שימו דגש רב על תזמונים והבנה של מקרי קצה, התממשקות זמנית בין סיגנלי הבקר והמידע שנמצא ברגע מסוים בנתיב הנתונים !!!

6. **<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>**

פתרון:

שימו לב שלתרגיל זה ישנן הרבה אפשרויות פתרון (יעילות יותר ופחות) אך בעיקר שונות. כלומר, מגוון גדול של פתרונות שיעבדו. הפתרון לא מגביל הרבה פתרונות שיתקבלו במחברות בחינה. **מכונת המצבים המתוארת היא מכונת MOORE כללית וסכמתית ולשם גיוון הקוד דווקא מתאר מכונת MEALY.** שימו לב שעבור מעגל ה data-path יכולתם להיעזר במעגל שניתן בהרצאה (כמובן שאתם לא מוגבלים לכך).



הקוד:

```
module StreamAddition #(parameter N_ADD = 16, N_BITS=4 )
( input clk,
 // Data signals
 input [N_BITS-1:0] IN,
 output reg [N_ADD-1:0] OUT,
 output reg [N_BITS-1:0] Counter,
 // Control signals (ctrl<->dpath)
 input Mode, //MODE=0 --> pass MODE=1 -->add
 input Lc, Lr, Reset
);
reg [N_ADD-1:0] out_tmp;

  always @(*)
 begin
    if (Mode)
    out_tmp = IN+OUT;
    else
    out_tmp = IN;
 end

 always @(posedge clk)
 begin
     if (Lr)
        OUT <= out_tmp;
 end

 always @(posedge clk)
 begin
     if (Lc)
     Counter <= IN;
     else if (Counter !=0)
        Counter <= Counter -1;


 end

 always @(posedge clk)
   if (Reset)
     OUT <= 0;
  endmodule


module FSM #(parameter N_ADD = 16, N_BITS=4 )
( input clk,
 // Control signals (ctrl->dpath)
 input  Reset,
 input [N_BITS-1:0] IN,
 // Control signals (dpath->ctrl)
 output reg Mode, //MODE=0 --> pass MODE=1 -->add
 output reg Lc, Lr, Ready,
 output reg Valid,
 input wire [N_BITS-1:0] Counter
);

localparam IDLE = 3'd0;
localparam START = 3'd1;
localparam SUM = 3'd2;
localparam DONE = 3'd3;
localparam HOLD = 3'd4;
localparam INVALID = 3'd5;

reg [2:0] state;

  always @(posedge clk)  //combinational and sequential in the same block
    if (Reset) //reset condition always first
      begin
       state <= IDLE; Lc <= 1'b1; Lr <= 1'b0; Mode <= 1'b0; Ready <= 0;
```

```
  end
else
  case ( state )
      IDLE : begin
             Ready <= 1'b0; Mode <= 1'b0;
                 if ( IN != 0 )  //reduction operator
                    if ((IN*(2**(N_BITS))-1)>(2**(N_ADD))-1)
                      begin
                       state <= HOLD;
                       Lc <= 1'b0; Lr <= 1'b0;
                      end
                    else
                      begin
                       state <= START;
                       Lc <= 1'b0; Lr <= 1'b1;
                      end
                 else
                   begin
                     Lc  <= 1'b1 ;  Lr <= 1'b0; Valid <= 1'b1;
                   end
             end
      START : begin
        if ( Counter === 4'b0001 )
                 begin
                    state <= DONE;
                 end
                else if ( Counter != 0 )
                  begin
                    state <= SUM;
                  end
                else
                begin
                    Mode  <= 1'b0; Lr    <= 1'b1;
```

```
                    Lc    <= 1'b0; Valid <= 1'b1;
                end
              end
      SUM :  begin
                 Mode <= 1'b1;
                 Lr   <= 1'b1;
                 Lc   <= 1'b0;
                 Valid<= 1'b1;
        if ( Counter === 4'b0001 )
                   state <= DONE;
                   else state <=SUM;
             end
      DONE : begin
                Mode  <= 1'b1;
                Lr    <= 1'b0;
                Valid <= 1'b1;
                Ready <= 1'b1;
                 if ( IN != 0 )
                   begin
                    state <= INVALID;   Lc <= 1'b0;
                   end
                 else
                   begin
                    state <= IDLE;   Lc <= 1;
                   end
             end
      HOLD : begin
                 Lr    <= 1'b0;
                 Valid <= 1'b0;
                 Ready <= 1'b0;
                  if ( Counter === 0 )
                   begin
                     if (IN === 0)
```

```
                        begin
                         state <= IDLE; Lc <= 1'b1;
                        end
                      else
                        begin
                         state <= INVALID; Lc <= 1'b0;
                        end
                    end
                  else
                    begin
                     Lc <= 1'b0;
                     state <= HOLD;
                    end
              end
        INVALID : begin
                    Ready   <= 1'b0;
                    Lr      <= 1'b0;
                    Lc      <= 1'b1;
                    Valid   <= 1'b0;
                    state <= (IN!==0) ? INVALID : IDLE;
              end
    endcase

endmodule
```

```
module TOP #(parameter N_ADD = 16, N_BITS=4 )
( input clk,
 // Data signals
 input [N_BITS-1:0] IN,
 output [N_ADD-1:0] OUT,
 input  Reset,
 output Ready,
 output Valid
);
wire [N_BITS-1:0] Counter;
wire Mode, Lc, Lr;

StreamAddition #( N_ADD , N_BITS ) StAdd
  ( .clk(clk), .IN(IN), .OUT(OUT), .Counter(Counter), .Mode(Mode), .Lc(Lc), .Lr(Lr), .Reset(Reset) );

FSM #( N_ADD , N_BITS ) fsm
  ( .clk(clk), .Reset(Reset), .IN(IN), .Mode(Mode), .Lc(Lc), .Lr(Lr), .Ready(Ready), .Valid(Valid), .Counter(Counter) );

endmodule
```