שאלה 1 (60 נק')

במעבד MULTI CYCLE MIPS נדרש לממש שתי פקודת חדשות מסוג R-type.
הראשונה נקראת (load word register) **lwr**. פורמט הפקודה הוא **lwr $R1 , $R2, $R3** , הקוד שלה **OPCODE=36** ופעולתה כדלהלן:

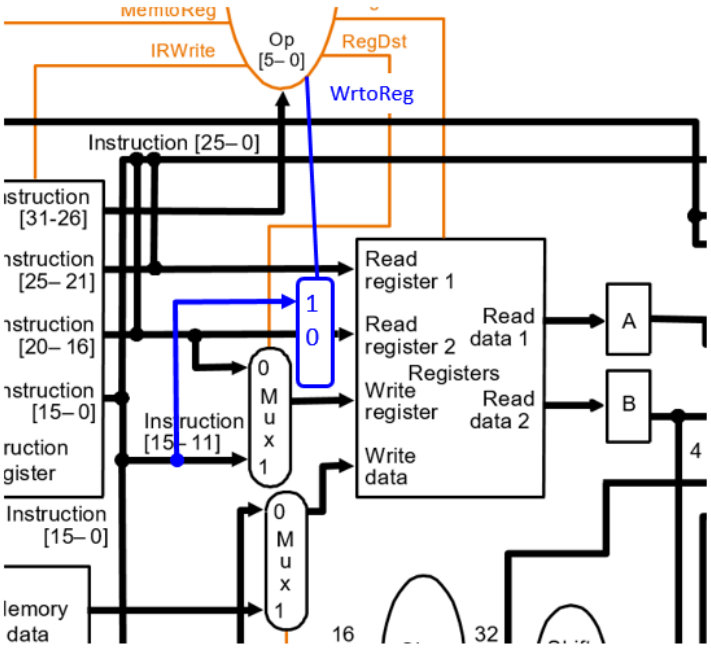
- קריאת כתובות זיכרון המאוכסנות באוגרים R2 ו R3
- קריאת מילה מהזיכרון מכתובת סכום R2 ו R3
- איכסון הערך שנקרא ב R1.

השניה נקראת (store word register) **swr**. פורמט הפקודה הוא **swr $R1 , $R2, $R3** , הקוד שלה **OPCODE=44** ופעולתה כדלהלן:

- קריאת כתובות זיכרון המאוכסנות באוגרים R2 ו R3
- קריאת הערך המצוי ב R1.
- כתיבת ערך R1 בזיכרון בכתובת סכום R2 ו R3

נדרש לממש את הפקודות במעבד שנלמד בקורס והתרשים שלו מופיע למטה.

א. (20 נק') לכל אחת משתי הפקודות **lwr** ו **swr** האם נדרשים שינויים / עדכונים במסלול הנתונים (DATA PATH)? אם לא נמקו מדוע. אם כן עדכנו בתרשים את הנדרש. הסבירו את המימוש בפירוט באופן מילולי וכן על גבי הסכמה המצורפת.



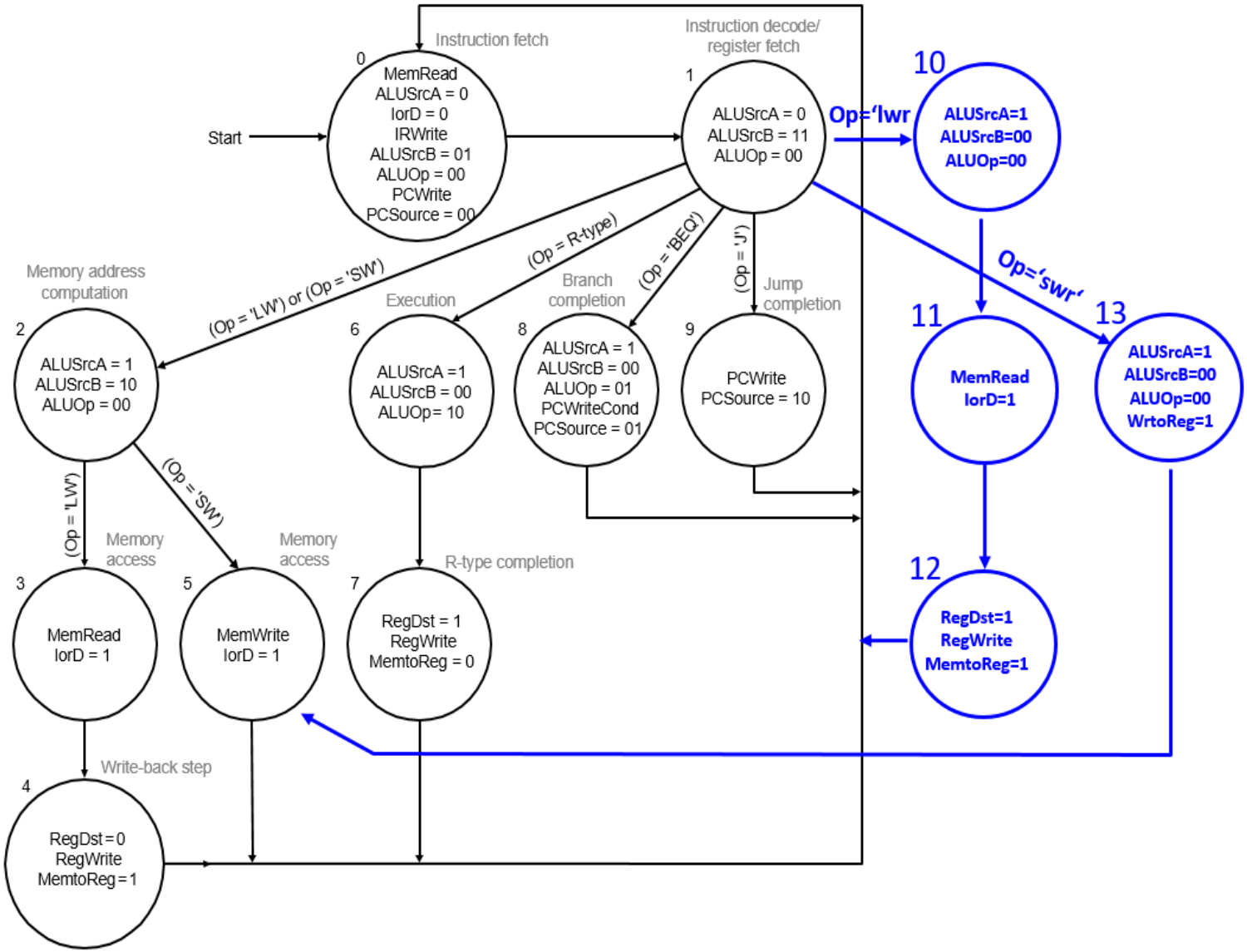
lwr: (10 נק') איננה דורשת שום שינוי במסלול הנתונים. ראשית כל חישוב כתובת הזיכרון הוא סכום של שני אופרנדים המצויים ברגיסטרים, בדיוק כמו כל פקודה מסוג R-type. ההמשך שבו נקרא ערך מהזכרון ומאוכסן ב memory data register גם הוא זהה לגמרי לפקודת lw וסיום הפקודה שהוא כתיבה לרגיסטר היעד זהה לזה של פקודת R-type.

swr: (10 נק') כאן כבר נדרש שינוי במסלול הנתונים משום שבפקודת sw מספר הרגיסטר ממנו נקרא הערך שיכתב לזכרון מצוי ב IR[20-16] ואילו בפקודת R-type הוא נמצא ב IR[15-11]. נדרש לכן בורר 2:1 בכניסה Read register 2 שב Register File.

ב. (20 נק') האם נדרשים שינויים / עדכונים בבקר (CONTROLLER)? אם לא נמקו מדוע. אם כן עדכנו בדיאגרמת המצבים את הנדרש, השתמשו במצבים הקיימים ככל שאפשר, ובהנחה שמימוש הבקר משתמש במיקרוקוד, המנעו מתוספת Dispatch ROM נוסף. הסבירו את המימוש בפירוט באופן מילולי וכן על גבי הסכמה המצורפת.

lwr: (10 נק') כל פקודה חדשה מחייבת תוספת למכונת המצבים של הבקר. בדומה לפקודת lw, נדרש מחזור (מצב) לחישוב כתובת הזיכרון, מחזור לקריאה מהזיכרון ומחזור write back לכתיבה ל Register File. אלה שלשת המצבים 10, 11 ו 12, בהתאמה. לפקודה זו לא נדרשים אותות בקרה חדשים אלא שבמצב write back ערכי אותות הבקרה צריכים לדאוג לכך שכתובת האוגר לכתיבה תהיה כמו ב R-type, ואילו מקור הערך הנכתב יהיה כמו בפקודת lw. ערכי אותות הבקרה מופיעים בדיאגרמה.

swr: (10 נק') מצבי הפקודה דומה מאד למצבי פקודת sw פרט לעובדה שמאחר ונוסף בורר 2:1 בכניסה Read register 2 שב Register File, נדרש אות בקרה נוסף שנקרא WrtoReg. המצב 13 אחראי לחישוב כתובת הזיכרון ואילו לגישה לזכרון ניתן להשתמש במצב 5 של sw.


Start
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Instruction fetch
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
Instruction decode/
register fetch
Op='lwr
10
ALUSrcA=1
ALUSrcB=00
ALUOp=00
Op='swr'
11
MemRead
IorD=1
13
ALUSrcA=1
ALUSrcB=00
ALUOp=00
WrtoReg=1
12
RegDst=1
RegWrite
MemtoReg=1
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
Execution
6
ALUSrcA =1
ALUSrcB = 00
ALUOp= 10
Branch
completion
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
Jump
completion
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
3
MemRead
IorD = 1
Memory
access
5
MemWrite
IorD = 1
R-type completion
7
RegDst = 1
RegWrite
MemtoReg = 0
Write-back step
4
RegDst = 0
RegWrite
MemtoReg = 1

מימוש מנגנון הבקרה במעבד נעשה בעזרת מיקרוקוד ושני Dispatch ROM המתוארים למטה.

ג. (20 נק') בהתאם לתשובה בסעיף ב', האם נדרש לשנות משהו בטבלאות Dispatch ROM הקיימים ובטבלת ערכי האות **AddrCtl**? אם לא נמקו מדוע. אם כן עדכנו את תרשים מימוש הבקר והטבלאות בהתאם.

<u>פתרון אפשרי:</u> בפתרון זה אין שינוי בקידוד המצבים הקיימים. מאחר ונוספו שתי פקודות אזי Dispatch ROM 1 המשמש את שלב ה Instruction Decode צריך לתמוך בפקודות החדשות כמתואר בטבלה. Dispatch ROM 2 נשאר ללא שינוי.

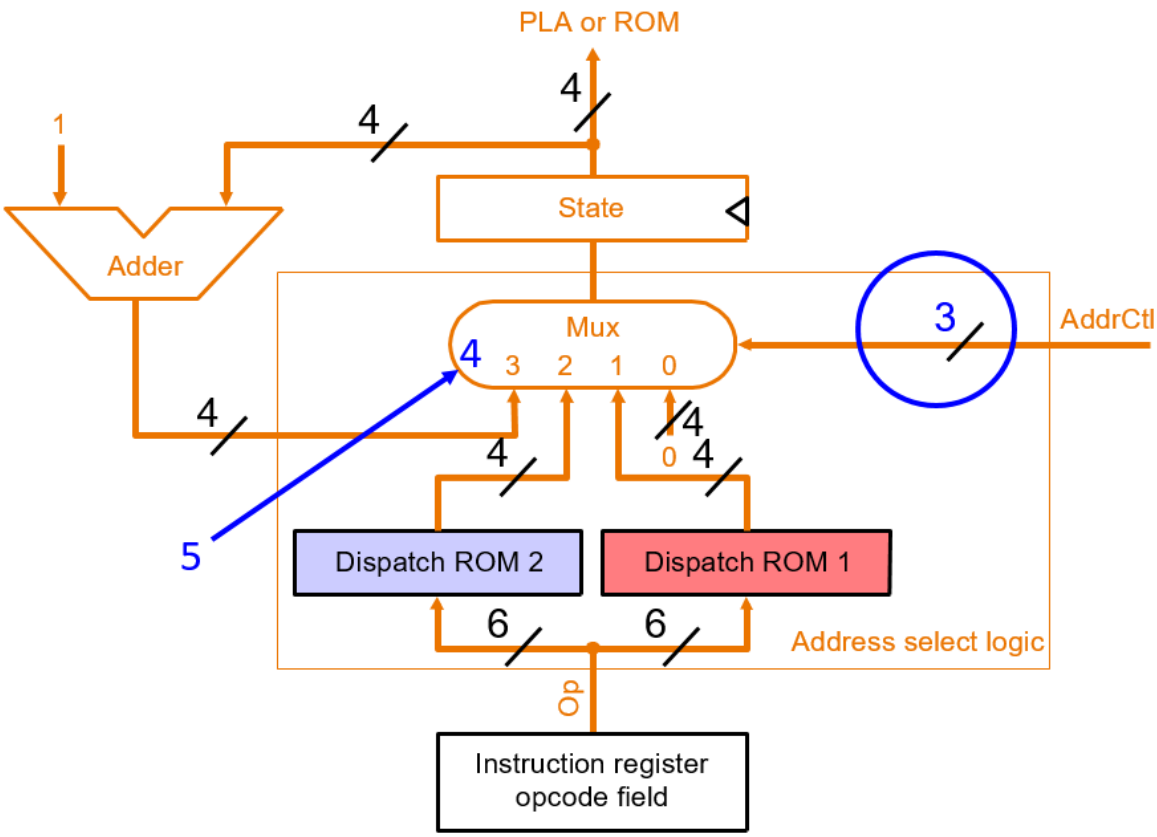
במכונת המצבים החדשה נוסף מעבר בפקודה **swr** ממצב 13 למצב 5, מעברים הנשלטים ע"י האות **AddrCtl.**

הבורר 4:1 שקובע את עדכון ה אוגר **State** (Microprogram Counter) הופך לבורר 5:1 ועלכן נדרשות לאות **AddrCtl** 3 סיביות.

הטבלה המתארת את האופן בו נקבעים מעברי המצבים של האוגר **State** צריכה לתמוך במעברי המצבים המתחייבים ממימוש שתי הפקודות החדשות.

Dispatch ROM 1		
Op	**Name**	**Value**
0	R-type	6
2	jmp	9
4	beq	8
35	lw	2
43	sw	2
36	**lwr**	**10**
44	**swr**	**13**

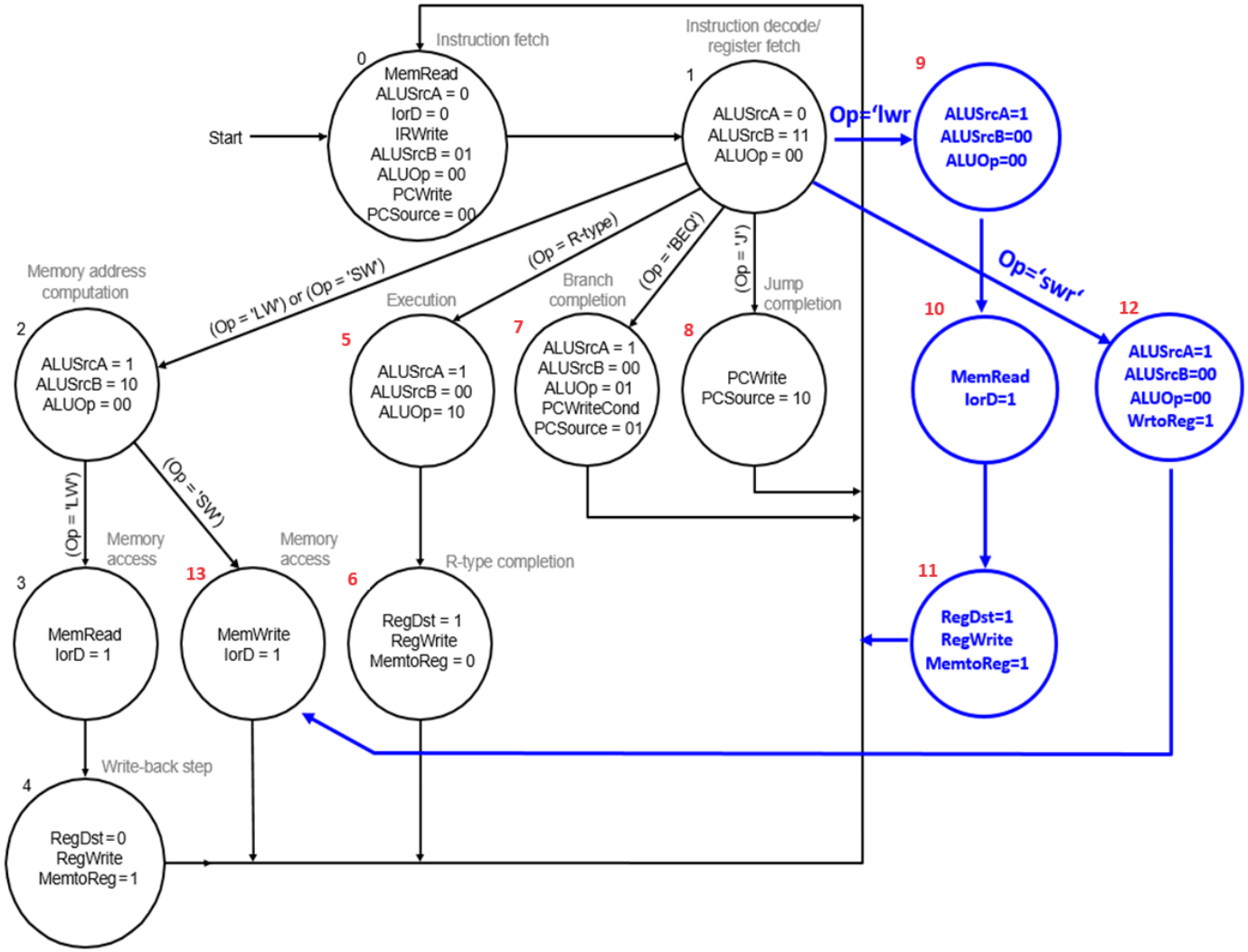
State number	Address control action	Value of AddrCtl
0	Increment	3
1	Use dispatch ROM 1	1
2	Use dispatch ROM 2	2
3	Increment	3
4	Go to state 0	0
5	Go to state 0	0
6	Increment	3
7	Go to state 0	0
8	Go to state 0	0
9	Go to state 0	0
10	**Increment**	**3**
11	**Increment**	**3**
12	**Go to state 0**	**0**
13	**Go to state 5**	**4**



<u>פתרון אפשרי אחר:</u> פתרון זה פשוט יותר ואינו מצריך הוספת סיבית לאות הבקרה **AddrCtl** אבל מצריך שינוי בקידוד המצבים הקיימים.

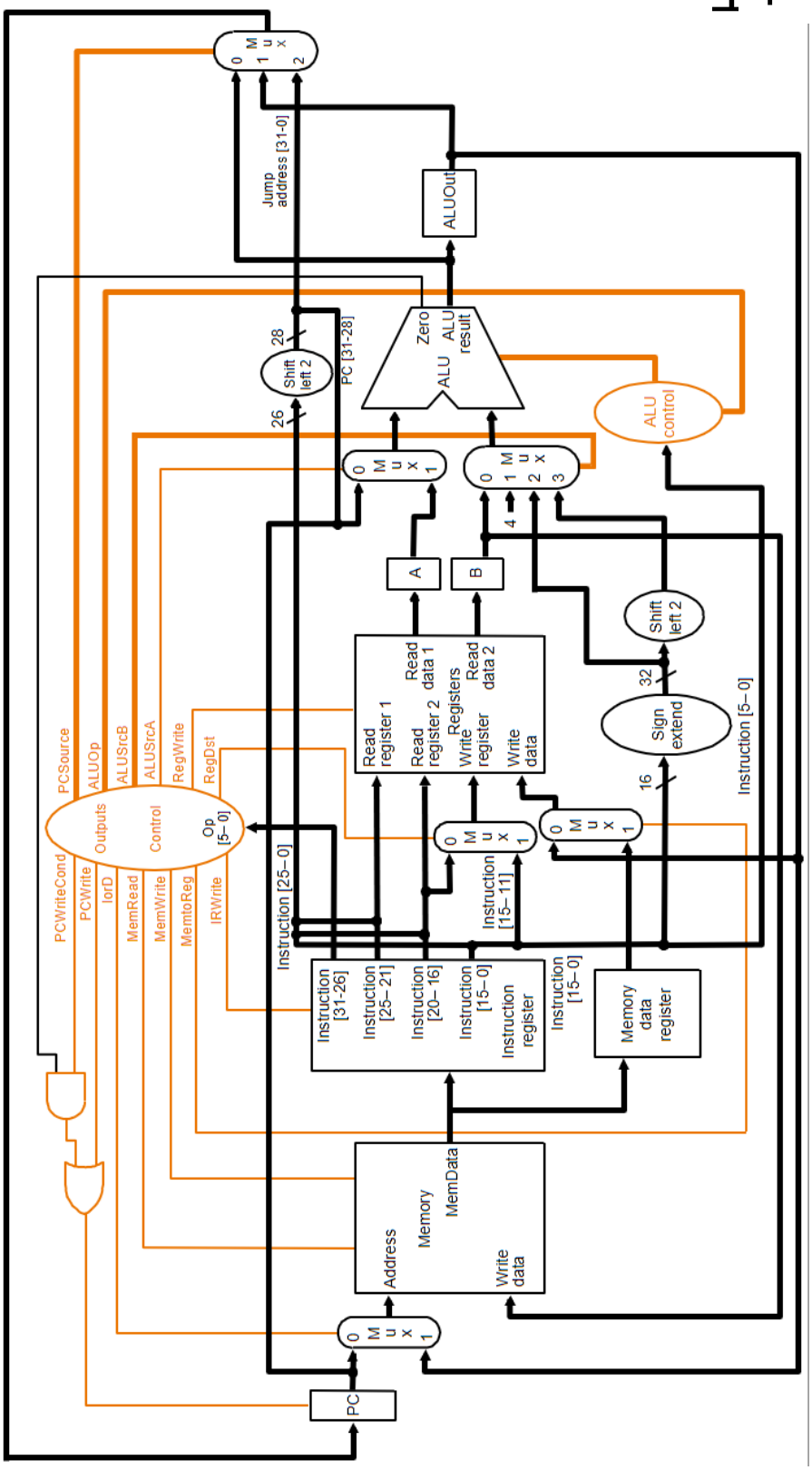
Dispatch ROM 2		
Op	**Name**	**Value**
35	Lw	3
43	Sw	13

Dispatch ROM 1		
Op	**Name**	**Value**
0	R-type	5
2	jmp	8
4	beq	7
35	lw	2
43	sw	2
36	lwr	9
44	swr	12

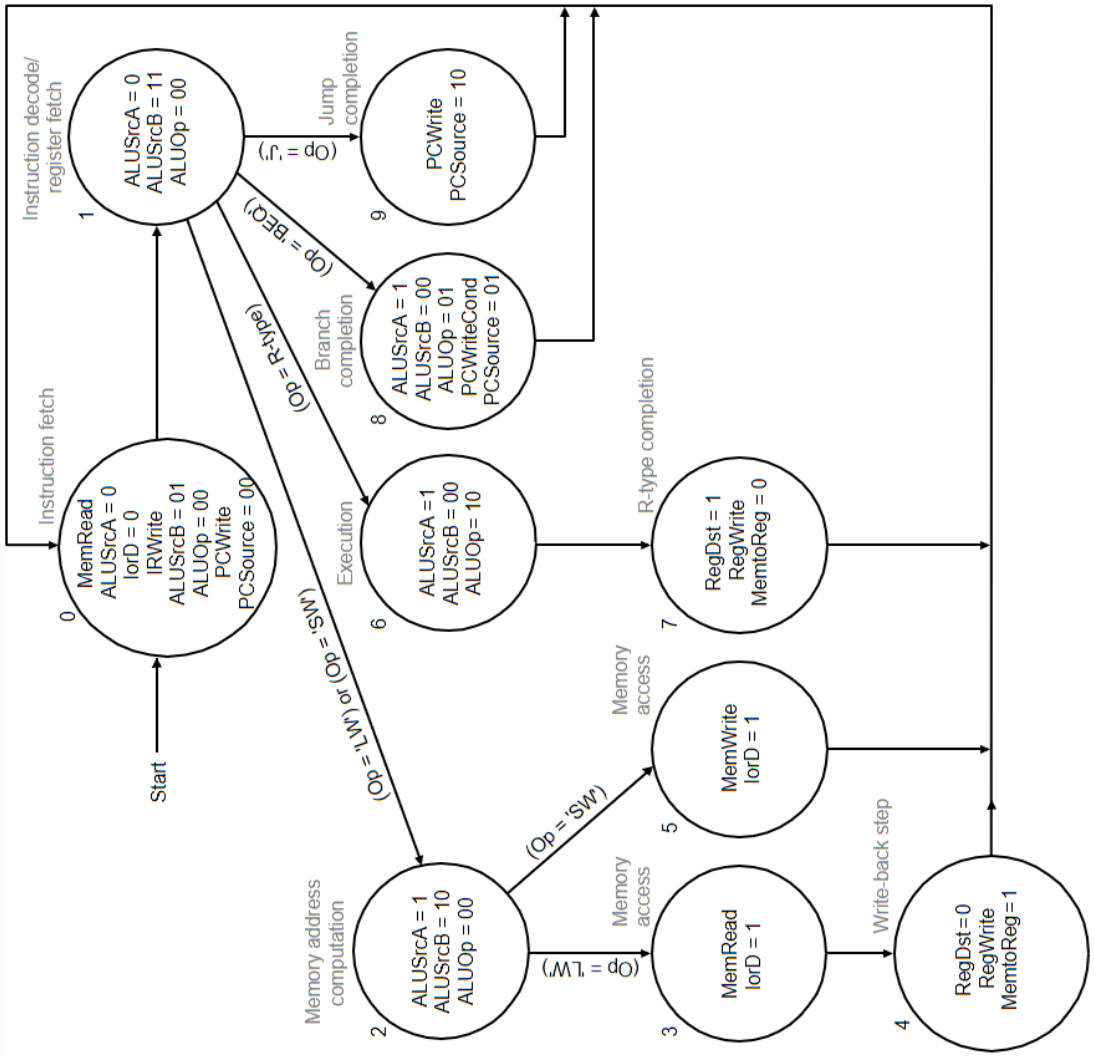

Start
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Instruction fetch
Instruction decode/
register fetch
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
Op='lwr
9
ALUSrcA=1
ALUSrcB=00
ALUOp=00
Op='swr'
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
Execution
5
ALUSrcA =1
ALUSrcB = 00
ALUOp= 10
Branch
completion
7
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
Jump
completion
8
PCWrite
PCSource = 10
10
MemRead
IorD=1
12
ALUSrcA=1
ALUSrcB=00
ALUOp=00
WrtoReg=1
(Op = 'LW')
(Op = 'SW')
Memory
access
3
MemRead
IorD = 1
Memory
access
13
MemWrite
IorD = 1
R-type completion
6
RegDst = 1
RegWrite
MemtoReg = 0
11
RegDst=1
RegWrite
MemtoReg=1
Write-back step
4
RegDst = 0
RegWrite
MemtoReg = 1

State number	Address control action	Value of AddrCtl
0	Increment	3
1	Use dispatch ROM 1	1
2	Use dispatch ROM 2	2
3	Increment	3
4	Go to state 0	0
5	Increment	3
6	Go to state 0	0
7	Go to state 0	0
8	Go to state 0	0
9	Increment	3
10	Increment	3
11	Go to state 0	0
12	Increment	3
13	Go to state 0	0

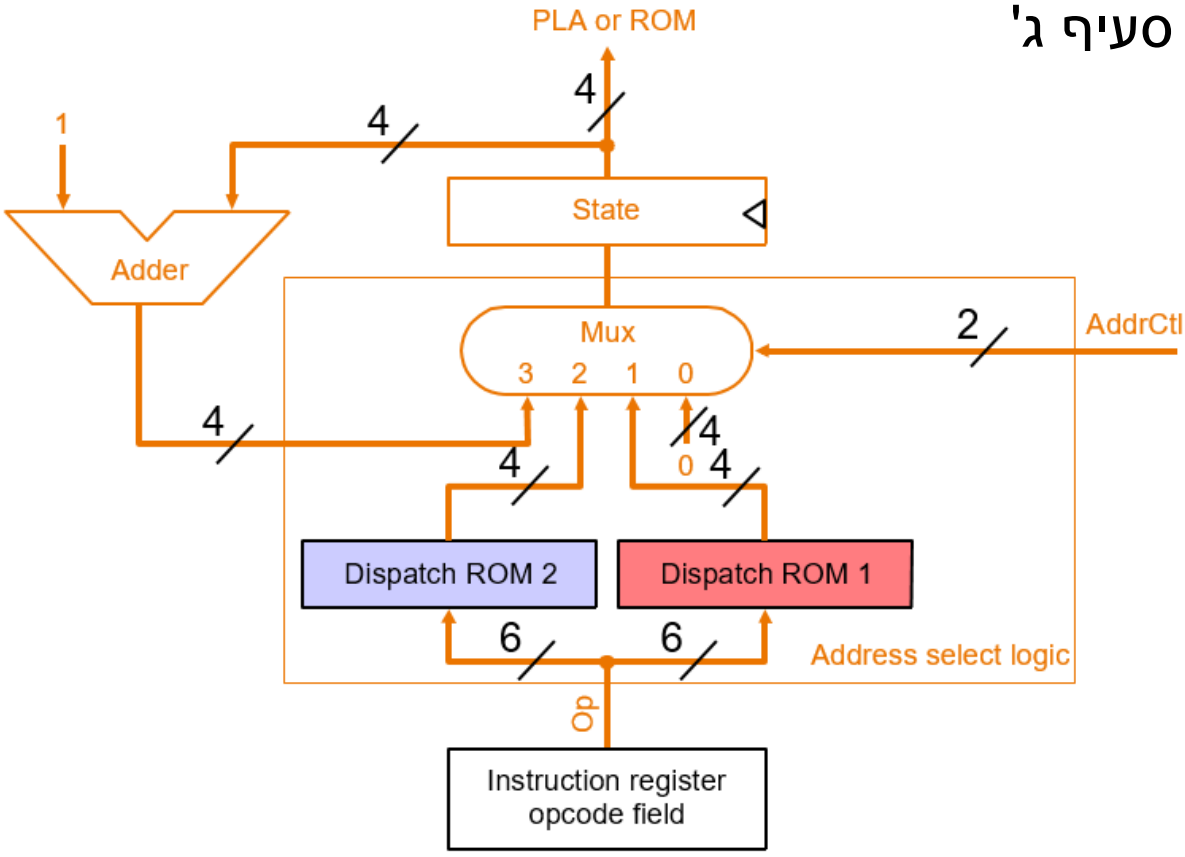
סעיף א' + ב'


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

סעיף ב'


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

סעיף ג'



Dispatch ROM 2		
Op	**Name**	**Value**
35	**Lw**	**3**
43	**Sw**	**5**

Dispatch ROM 1		
Op	**Name**	**Value**
0	**R-type**	**6**
2	**jmp**	**9**
4	**beq**	**8**
35	**lw**	**2**
43	**sw**	**2**

סעיף ג'

State number	Address control action	Value of AddrCtl
0	Increment	3
1	Use dispatch ROM 1	1
2	Use dispatch ROM 2	2
3	Increment	3
4	Go to state 0	0
5	Go to state 0	0
6	Increment	3
7	Go to state 0	0
8	Go to state 0	0
9	Go to state 0	0

<u>שאלה 2</u> (60 נק')

נתונה התכנית שלהלן המיועדת ל MIPS ומשתמשת ב delay slots.

```
loop:
    LW    R4,   0(R3) // 1
    ADDI  R5,   R4,   0 // 2
    SLL   R5,   R5,   1 // 3
    SLR   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:

BNEZ קופצת לכתובת אם האוגר שונה מאפס.

SLL - Shift Left Logical

SLR - Shift Right Logical

פקודות ה- shift אינן ציקליות, כלומר הזזה ימינה זורקת LSB ומאפסת MSB והזזה שמאלה מאפסת LSB וזורקת MSB.

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

א. (10 נק') מה מבצעת התכנית הנ"ל? מהו ערכו של R2 בעת היציאה מהלולאה?

התוכנית מקבלת מערך באורך $n + 1$ בכתובת p וסופרת כמה מאיבריו שליליים. בדיקת הסימן נעשית ע"י הזזה שמאלה ואחר כך ימינה. אם המספר אי שלילי מלכתחילה, מקרבה שבו MSB=0, מספר אי שלילי איננו משתנה. לו המספר שלילי, מקרה שבו מלכתחילה MSB=1, ההזזות תהפוכנה אותו לאי שלילי. בסיום התוכנית R2 מכיל את מספר האיברים השליליים.

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

ב. (15 נק') האם התכנית תתבצע באופן תקין? נמקו את תשובתכם באופן מפורט. אם התשובה שלילית ציינו במדויק אילו תיקונים נדרשים בתכנית ובאיזה מקומות.

התוכנית תבצע באופן שגוי משום שלפי התרשים הבדיקה לקיום קפיצה מותנית מתבצעת בשלב ה MEM של הצינור, ובצינור כבר נמצאות שלש פקודות נוספות. רק פקודה 6 (קידום אינדקס במערך) הינה בלתי מותנית גם לו הקפיצה אמורה הייתה להתבצע (taken). בכדי שהתוכנית תתבצע נכון יש לדאוג ש:

1. הפקודה 7 תתבצע רק אם הקפיצה 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 לבדיקת איברי המערך! בכדי למנוע זאת יש להוסיף שתי פקודות NOP אחרי פקודה 9.

ג. (15 נק') 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 התרשים והסבירו כיצד השינוי פותר את הבעיה.

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, דבר המצריך הוספת משווה (comparator) במוצא ה register file.
2. הקדמת חישוב כתובת הקפיצה משלב ה EXE לשלב ה ID, דבר המחייב הוספת adder לצורך חישוב כתובת הקפיצה.

כמובן ששני השינויים הנ"ל מחייבים שימוש במנגנון ה forwarding.
אסור ליצור בטעות מנגנון של flash ב IF למקרה שהקפיצה taken, משום שהדבר יוביל להשמדה מוטעית של הפקודה המצויה ב delay slot שאמורה להתבצע תמיד.

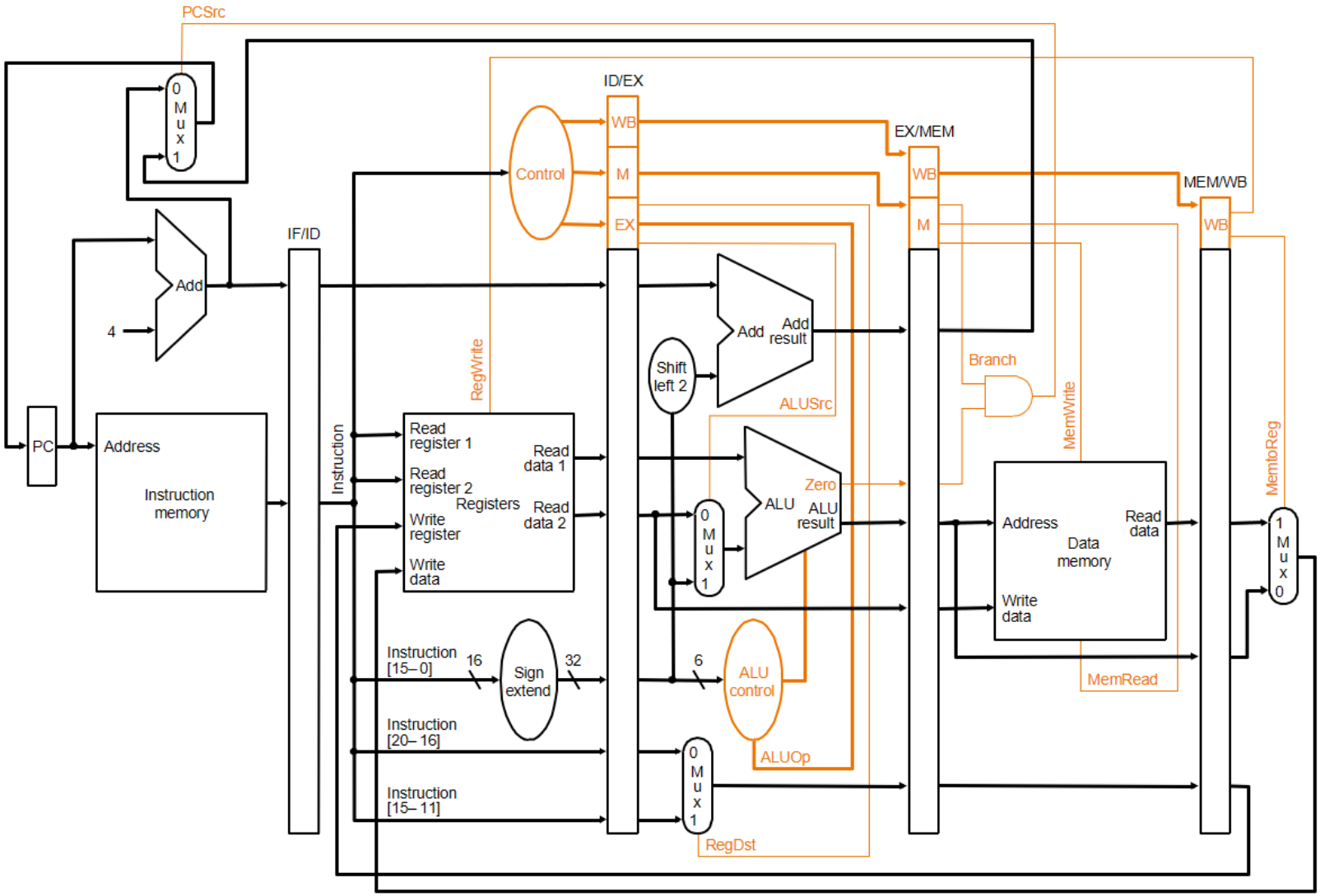
ד. (20 נק') נתון מערך בין שני איברים כשהראשון הוא 6 והשני 11-. רשמו בטבלה את כל פקודות התכנית ושלביהן (IF, ID, X-EXE, M-MEM, W-WB) המבוצעות במהלך 21 מחזורי

השעון הראשונים. בעמודה INSTR שבטבלה יש לרשום את מספר הפקודה כפי שמופיע בקוד.

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 אי שלילי), 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.

בנוסף, נשים לב שפקודת הקפיצה 5 זקוקה בשלב ה ID לערך של R5 שמחושב בפקודה 4 שנמצאת בשלב EXE . במדה ונשתמש ביחידת ה forwarding אפשר לקחת את הערך של R5 מתוך הרגיסטר EXE/MEM במחזור השעון הבא. הדבר מחייב ביצוע stall לפקודת הקפיצה, השהיית שעון לפקודות 5 ו 6. ניתן להימנע מ stall ע"י הוספת יחידת forwarding נוספת לשלב ה EXE שמעבירה את תוצאת ה ALU לשלב ה ID באותו מחזור שעון. פתרון כזה אמנם לא ימנע את ה stall, אבל עלול לגרום למסלול קריטי משילוב ALU, יחידת forwarding ו comparator. המחיר יהיה הגדלת זמן מחזור השעון.

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	ID													
6							IF	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	ID			
6																	IF	IF	ID	X	M
7																			IF	ID	X


PCSrc
0
M
u
x
1
Add
4
PC
Address
Instruction
memory
IF/ID
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]
ID/EX
Control
WB
M
EX
Shift
left 2
Add
Add
result
ALUSrc
0
M
u
x
1
ALU
Zero
ALU
result
6
ALU
control
ALUOp
RegDst
EX/MEM
WB
M
Branch
MemWrite
Address
Data
memory
Write
data
Read
data
MemRead
MEM/WB
WB
MemtoReg
1
M
u
x
0