אוניברסיטת בר-אילן (ע"ר)
הפקולטה להנדסה
רמת-גן 52900

BAR-ILAN UNIVERSITY (RA)
Faculty of Engineering
Ramat-Gan 52900, Israel

Tel: 03-5317722

engbi@mail.biu.ac.il

תכן לוגי

תשע"ט סמסטר קיץ מועד ב'

83-253

מרצה: פרופ' שמואל וימר

מתרגל: מר בנימין פרנקל

- **יש לקרוא היטב את ההוראות.**
- **חובה לענות על כל השאלות.**
- **סך הנקודות בבחינה הוא 110 אך הציון הסופי בבחינה לא יעלה על 100.**
- **יש לנמק את כל תשובותיכם.**
- יש לשרטט באופן ברור, להוסיף הסבר מילולי, ובמידת הצורך לצייר את החלק הרלוונטי במחברת הבחינה.
- יש להקפיד על כתב יד קריא!
- **חומר עזר מותר בשימוש**: מחשבון, ספר הקורס ושקפים מההרצאות בלבד (כולל הערות על גביהם).
- **משך הבחינה:** שלוש שעות.
- **יש לצרף את שאלוני הבחינה למחברת!**

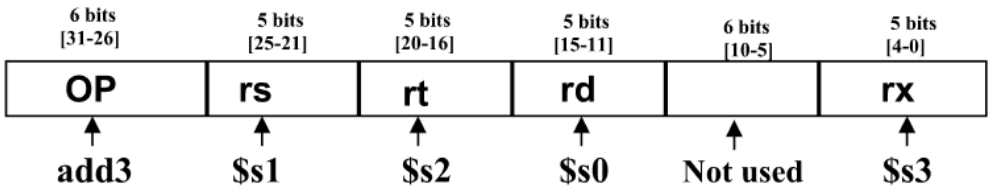
בהצלחה!

<u>שאלה מספר 1</u> (40 נקודות):

בשאלה זו נרצה לשנות את מבנה ה- Multi cycle MIPS כך שיתמוך בפקודה חדשה:

הפקודה **add3** מבצעת סכימה של 3 מספרים האגורים בשלושה רגיסטרים, ושומרת את הסכום ברגיסטר רביעי. הפקודה נכתבת באופן הבא: **add3 $s0, $s1, $s2, $s3**,

כאשר השדות בפקודה מוגדרים כך:



א. עליכם להוסיף ו/או לשנות את מסלול הנתונים המתואר בתרשים מספר 1 כך שהחומרה תתמוך בביצוע הפקודה. אין לשנות את ה- ALU. יש להסביר במחברת באופן מילולי ומפורט את כל השינויים שנעשו על גבי הסכמה. אם אין צורך בשינוי של מסלול הנתונים יש להסביר היטב כיצד החומרה מבצעת את הפקודה בכל זאת.

ב. שנו את מכונת המצבים של הבקר בתרשים מספר 2 כך שהפקודה החדשה תתבצע במספר מינימלי של מחזורי שעון. יש להעדיף שימוש במצבים קיימים. אם אתם מוסיפים או משנים מצבים יש לרשום בתוכם את ערכי אותות הבקרה וה- enables הרלוונטיים. כמה מחזורי שעון לוקח ביצוע הפקודה?

5 מחזורי שעון.

ג. במקום לממש את הפקודה add3 באופן ישיר בחומרה, הועלה רעיון לממש אותה בתכנה (software) כ- pseudo-instruction. רשום רצף פקודות מתוך כל הפקודות שנלמדו בקורס על מנת לממש את הפקודה add3.

```
add   $s0   $s1   $s2
add   $s0   $s0   $s3
```

ד. נניח שמימוש הפקודה add3 באופן תכנתי (software implementation) לוקח 8 מחזורי שעון בסה"כ, בעוד שמימוש הפקודה באופן חומרתי (hardware implementation) לוקח 5 מחזורי שעון, אבל מחייב להגדיל את מחזור השעון של המעבד ב- 10%. כמה פקודות add3 צריכה תכנית לבצע ביחס לשאר הפקודות בתכנית (באחוזים) כדי שהמימוש החמרתי יהיה כדאי מבחינת זמן ביצוע (execution time)? נתון כי זמן הביצוע הממוצע עבור כל שאר הפקודות הוא 4 מחזורי שעון.

זמן הביצוע של התכנית הוא:

$$Execution\ time = IC \times CPI \times CCT$$

נסמן את המרכיבים של זמן הביצוע של המימוש ב- software כך: $IC_s \times CPI_s \times CCT_s$

ובאופן דומה, נסמן את המרכיבים של זמן הביצוע של המימוש ב- hardware:

$\mathrm{IC}_h \times CPI_h \times CCT_h$.

נסמן ב- I את מספר הפקודות שאינן add3, ונסמן ב- A את מספר הפקודות שהן add3.

על פי נתוני השאלה:

$$CPI_s = 4 \times I + 8 \times A$$

$$CPI_h = 4 \times I + 5 \times A$$

נבדוק מתי זמן הביצוע משתווה:

$$\mathrm{IC}_s \times CPI_s \times CCT_s = \mathrm{IC}_h \times CPI_h \times CCT_h$$

$$(I + A) \times (4 \times I + 8 \times A) \times CCT_s = (I + A) \times (4 \times I + 5 \times A) \times 1.1 \cdot CCT_s$$

$$(4 \times I + 8 \times A) = (4 \times I + 5 \times A) \times 1.1$$

$$2.5A = 0.4I$$

$$\frac{A}{I} = \frac{0.4}{2.5} = \frac{4}{25} = 16\%$$

ולכן, כדי שיהיה כדאי להשתמש בפתרון ה- hardware צריך להתקיים שהיחס בין פקודות ה-add3 שמתבצעות במהלך התכנית לבין שאר הפקודות יהיה גדול מ- 16%.

<u>שאלה מספר 2</u> (50 נקודות):

בשאלה זו נעסוק בטיפול ב- Hazards הקיימים בארכיטקטורת Pipelined MIPS.

נתון רצף פקודות האסמבלי הבא:

```
Loop:   lw      $t0     0($a0)
        sub     $t1     $t0     $a2
        lw      $t0     0($a1)
        add     $t1     $t1     $t0
        sw      $t1     0($a0)
        xor     $t1     $t1     $t0
        sw      $a2     0($t1)
        addi    $a0     $a0     4
        addi    $a1     $a1     4
        addi    $a3     $a3     -1
        bne     $a3     $zero   Loop
        addi    $a0     $zero   0
        addi    $a1     $zero   0
        addi    $a2     $zero   0
```

התכנית רצה על מעבד בארכיטקטורת Pipelined MIPS הפועל בתדר f_1, כאשר שלב ה- Instruction Fetch של הפקודה הראשונה (lw) מתבצע במחזור שעון מספר 1.

הניחו כי למעבד קיימת יחידת forwarding המתוארת בתרשים מספר 3, וכן קיימת יחידת hazard detection היודעת להכניס bubble במקרה הצורך (היא אינה מתוארת בתרשים מספר 1, אך היא בהחלט קיימת).

עוד הניחו כי שלב ה- branch resolution נמצא בשלב ה- Memory (לא מתואר בתרשים מספר 3, אך קיים), כאשר המעבד ממשיך להכניס ל-pipe באופן רציף את הפקודות הבאות אחרי כל פקודת branch. אם מתברר שתנאי הקפיצה התקיים – החומרה יודעת להשמיד את הפקודות שנכנסו בטעות למעבד.

א. חשבו מהו מספר מחזורי השעון הדרוש על מנת להריץ את התכנית על המעבד הנ"ל. קחו בחשבון את כל ה- stalls וה- flushes. הניחו כי $a3 מאותחל לערך 100.

האיטרציה השנייה תוכל לעשות IF לפקודת ה- lw הראשונה שלה במחזור שעון מספר 17, ולכן, באופן אפקטיבי, כל איטרציה לוקחת 16 מחזורי שעון. עבור 100 איטרציות ידרשו 1600 מחזורי שעון, כאשר על מנת להשלים את האיטרציה האחרונה יש צורך בעוד 4 מחזורי שעון. לכן, דרושים 1604 מחזורי שעון על מנת להריץ את התכנית על המעבד הנ"ל.

ב. מלאו את הדיאגרמה הבאה, המתארת את התקדמות 14 הפקודות הראשונות הנכנסות למעבד בתוך שלבי ה- pipeline השונים כתלות במספר מחזור השעון.

# C.C / Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw	IF	ID	EX	M	WB															
sub		IF	ID	--	EX	M	WB													

lw			IF	--	ID	EX	M	WB											
add				--	IF	ID	--	EX	M	WB									
sw						IF	--	ID	EX	M	WB								
xor							--	IF	ID	EX	M	WB							
sw									IF	ID	EX	M	WB						
addi										IF	ID	EX	M	WB					
addi											IF	ID	EX	M	WB				
addi												IF	ID	EX	M	WB			
bne													IF	ID	EX	M	WB		
addi														IF	ID	EX	--	--	
addi															IF	ID	--	--	--
addi																IF	--	--	--	--

- היעזרו בתרשים מספר 3 לצורך מענה על הסעיף הבא:

ג. בסעיף זה עליכם לנתח את פעולת ה- Forwarding Unit במהלך ריצת התכנית. בטבלה הבאה עליכם לרשום עבור כל מחזור שעון את הערכים של קווי הבקרה המנתבים את זרימת המידע בשלב ה- Execution. אם ישנם Don't-cares ציינו זאת בטבלה!

Clock Cycle	ForwardA	ForwardB	ALUsrc	RegDst
3 = lw	0	x	1	0
4 = bubble	x	x	x	x
5 = sub	1	0	0	1
6 = lw	0	x	1	0
7 = bubble	x	x	x	x
8 = add	0	1	0	1
9 = sw	0	2	1	x
10 = xor	1	0	0	1
11 = sw	2	0	1	x
12 = addi	0	x	1	0
13 = addi	0	x	1	0
14 = addi	0	x	1	0
15 = bne	2	0	0	x
16 = addi	0	x	1	0
17 = bubble	x	x	x	x
18 = bubble	x	x	x	x

19 = lw	0	x	1	0
20 = bubble	x	x	x	x

ד. מהנדס צעיר, בוגר הקורס "תכן לוגי" באוניברסיטת בר-אילן, הציע לשנות את החומרה כך ששלב ה- branch resolution יהיה בשלב ה- Execution. בשל השינויים בחומרה, על המעבד לפעול כעת בתדר f_2 . האם הצעתו של המהנדס הצעיר יכולה לשפר את משך ריצת התכנית הנ"ל? אם כן, חשב מהו מספר מחזורי השעון הדרוש כדי להריץ את התכנית על החומרה החדשה, ומהו התנאי על תדר השעון החדש f_2 כדי שמשך ריצת התכנית אכן יהיה קצר יותר מאשר במעבד המקורי. אם הצעתו של המהנדס הצעיר אינה יכולה לשפר את משך ריצת התכנית, הסבר מדוע.

אכן, הצעתו של המהנדס הצעיר יכולה לשפר את זמן ביצוע התכנית, אך הדבר תלוי ביחס בין f_2 לבין f_1 . כאשר מתברר כי תנאי הקפיצה התקיים, החומרה החדשה תמזער את הפגיעה בביצועים כך שנאבד רק שני מחזורי שעון בכל פעם, ולא שלושה מחזורי שעון. תנאי הקפיצה מתקיים 99 פעמים במהלך ריצת התכנית, ולכן החומרה החדשה תריץ את התכנית ב- 1505 מחזורי שעון.

כדי שהדבר אכן ישפר את משך ריצת התכנית יש לדרוש כי: $\frac{1505}{f_2} < \frac{1604}{f_1}$, דהיינו:

$f_2 > \frac{1505}{1604} f_1$.

ה. מהנדס אחר, בוגר מכללה להנדסה, הציע הצעה אחרת, לפיה נוסיף חומרה למעבד כך ששלב ה- branch resolution יהיה בשלב ה- ID. בשל השינויים בחומרה, על המעבד לפעול כעת בתדר f_3 . האם הצעתו של המהנדס בוגר המכללה יכולה לשפר את משך ריצת התכנית הנ"ל? אם כן, חשב מהו מספר מחזורי השעון הדרוש כדי להריץ את התכנית על החומרה החדשה שהציע בוגר המכללה, ומהו התנאי על תדר השעון החדש f_3 כדי שמשך ריצת התכנית אכן יהיה קצר יותר מאשר בהצעות הקודמות. אם הצעתו של בוגר המכללה אינה יכולה לשפר את משך ריצת התכנית, הסבר מדוע.

הצעתו של בוגר המכללה איננה יכולה לגרום לתכנית הנתונה בשאלה להתבצע במספר מחזורי שעון קטן יותר מ- 1505, שהרי פקודת הקפיצה המותנית זקוקה לערך של $a3 המחושב בפקודה הקודמת (הנמצאת בשלב ה- Ex), ולכן יהיה צורך ב- stall של מחזור שעון אחד, ולכן אין כל יתרון בהצעה זו על פני הצעתו של המהנדס הצעיר מבר אילן מבחינת מספר מחזורי שעון. אדרבה, מהבחינה הזאת הצעתו של בוגר המכללה עוד דורשת מחזור שעון נוסף על פני הצעתו של המהנדס הצעיר, שכן נדרש stall גם באיטרציה האחרונה של הלולאה, ולכן ידרשו 1506 מחזורי שעון.

מה שכן, אם מתקיים: $f_3 > \frac{1506}{1604} f_1$ הרי שמשך ריצת התכנית יהיה יותר קצר מאשר זמן הריצה

על המעבד המקורי, ואם מתקיים: $f_3 > \frac{1506}{1505} f_2$ הרי שזמן ריצת התכנית יהיה יותר קצר מאשר זמן הריצה על-פי הצעתו של המהנדס הצעיר מהסעיף הקודם.

- סטודנט מצטיין מאוניברסיטת בר אילן הצליח לשפר את זמן ריצת התכנית, וזאת מבלי לשנות דבר בחומרה של המעבד המקורי (המתואר בתחילת השאלה), אלא ע"י שינוי סדר הפקודות של התכנית המקורית בלבד. היות ולא נעשה כל שינוי בחמרה, המעבד עדיין יכול לעבוד בתדר f_1 .

ו. כתבו את התכנית מחדש, באופן שיגרום לתכנית להתבצע בכמה שפחות מחזורי שעון. חשבו מהו מספר מחזורי השעון הדרוש כעת על מנת להריץ את התכנית שכתבתם על המעבד המתואר בתחילת השאלה. מהו ה- speedup המתקבל בעקבות שינוי סדר הפקודות שרשמתם (ביחס לזמן הריצה המקורי שמצאתם בסעיף א')?

```
Loop:  lw    $t0   0($a0)
       addi  $a3   $a3   -1
       sub   $t1   $t0   $a2
       lw    $t0   0($a1)
       addi  $a1   $a1   4
       add   $t1   $t1   $t0
       sw    $t1   0($a0)
       xor   $t1   $t1   $t0
       sw    $a2   0($t1)
       addi  $a0   $a0   4
       bne   $a3   $zero Loop
       addi  $a0   $zero 0
       addi  $a1   $zero 0
       addi  $a2   $zero 0
```

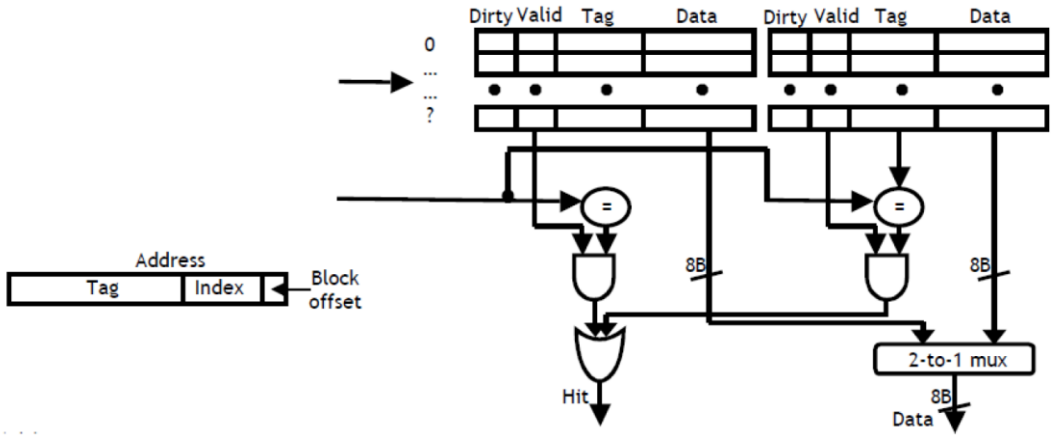
ע"י שינוי סדר הפקודות הנ"ל, נחסכו 2 מחזורי שעון בכל איטרציה, כך שבאופן אפקטיבי, כל איטרציה לוקחת 14 מחזורי שעון. לכן, נדרשים 1400 עבור 100 איטרציות, כאשר נדרשים עוד 4 מחזורי שעון על מנת להשלים את האיטרציה האחרונה. לכן, בסה"כ נדרשים 1404 מחזורי שעון על מנת להריץ את התכנית על המעבד המקורי. שיפור הביצועים הוא:

$$Speedup = \frac{1604}{1404} \simeq 1.14245$$

<u>שאלה מספר 3</u> (20 נקודות):

בשאלה זו נעסוק בניתוח של זיכרון מטמון.

נתונה מערכת זיכרון מטמון המתוארת בתרשים הבא:



נתון כי גודלה של כתובת הוא 32 סיביות, גודלו של כל בלוק הוא 8-Byte (כפי שגם ניתן לראות בתרשים), וכן כי גודלו של שדה ה- index הוא 10 סיביות. עוד נתון כי רזולוציית המיעון היא 1 Byte.

א. מהו גודלו של שדה ה- Tag?

מכיוון ששדה ה- offset הוא בגודל 3 סיביות, הרי ששדה ה- Tag הינו בגודל של:

$$32 - 10 - 3 = 19[bit]$$

ב. כמה מידע יכול להכיל זיכרון המטמון?

כל אינדקס מצביע ל- 2 בלוקים (2-way associative), ולכן יש $2^{10} \cdot 2 = 2^{11}[Blocks]$

כל בלוק מכיל 8 בתים, ולכן סה"כ מידע: $2^{11} \cdot 8 = 2^{14} = 16KB$

ג. מהו גודלו של זיכרון המטמון בביטים?

בכל בלוק, בנוסף ל- 64 סיביות מידע, יש גם 19 סיביות של ה- Tag, סיבית אחת של Valid וסיבית אחת של Dirty, ולכן, בסה"כ:

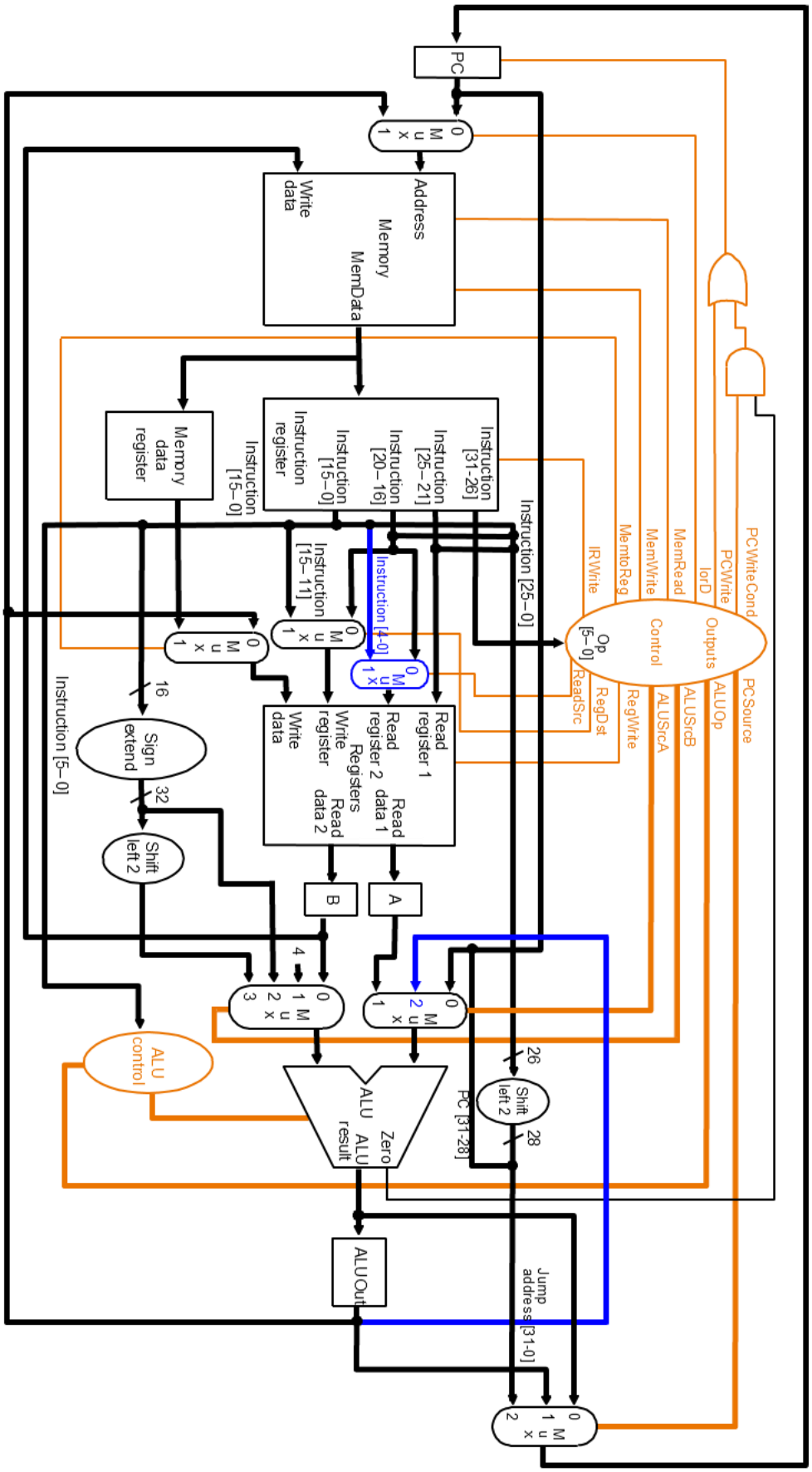
$$2^{11} \cdot [64 + 19 + 1 + 1] = 2048 \times 85 = 174,080[\text{bit}]$$

ד. נתון זיכרון מטמון - directly mapped cache בעל 4 בלוקים של 8 בתים כל אחד. עבור רצף הגישות המתואר בטבלה הבאה, ציינו לאלו גישות זיכרון היה miss/hit. כמו-כן, עבור גישות הזיכרון בהן היה hit ציינו איזו לוקליות באה לידי ביטוי, ועבור גישות הזיכרון בהן היה miss ציינו מהי הסיבה לכך.

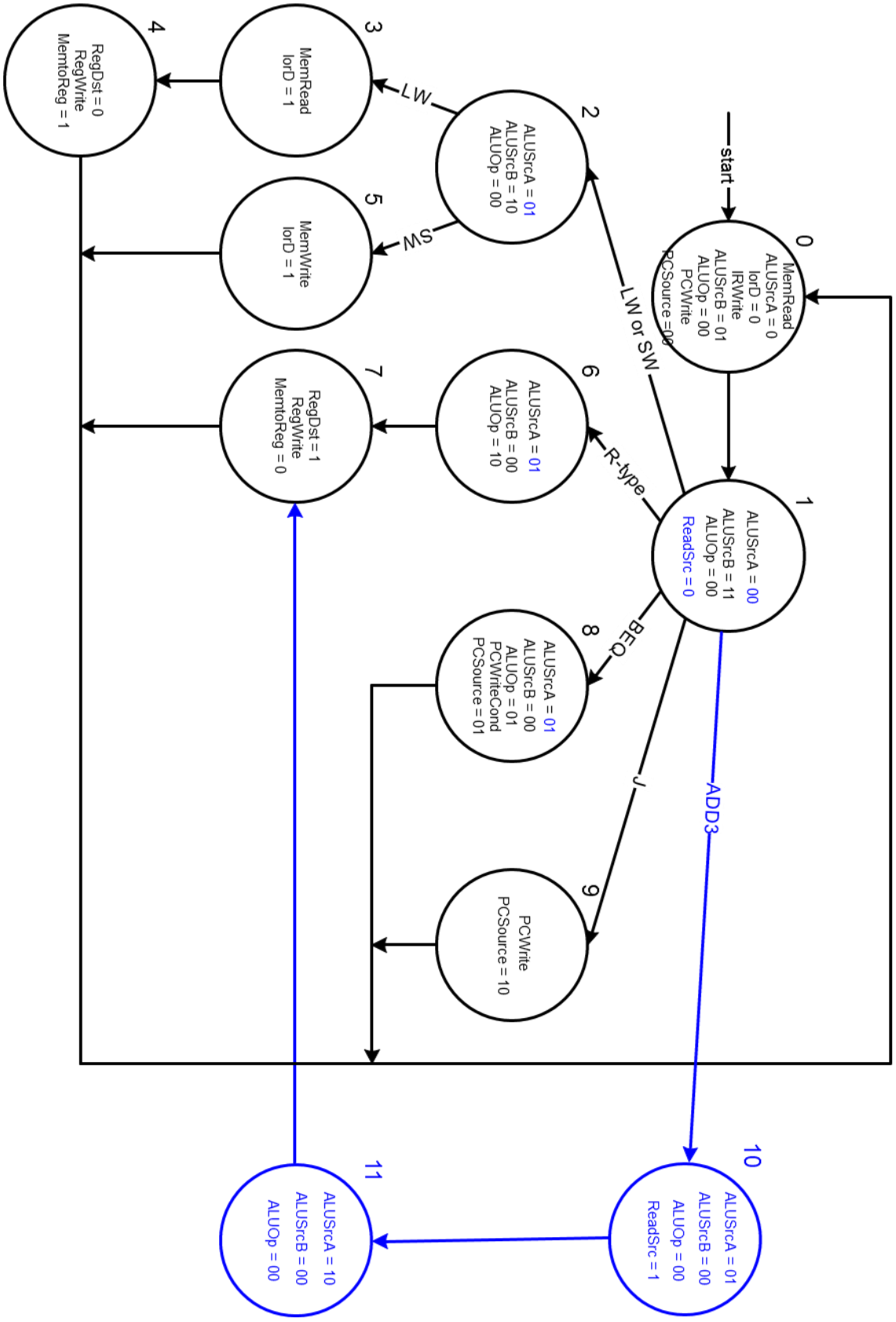
Binary Address	Miss/Hit	Reason
000000	Miss	Compulsory
010101	Miss	Compulsory
111111	Miss	Compulsory
010000	Hit	Spatial locality

011111	Miss	Conflict
000100	Hit	Spatial locality
111111	Miss	Conflict
110111	Miss	Conflict
000111	Hit	Spatial locality
111111	Hit	Temporal locality
011100	Miss	Conflict

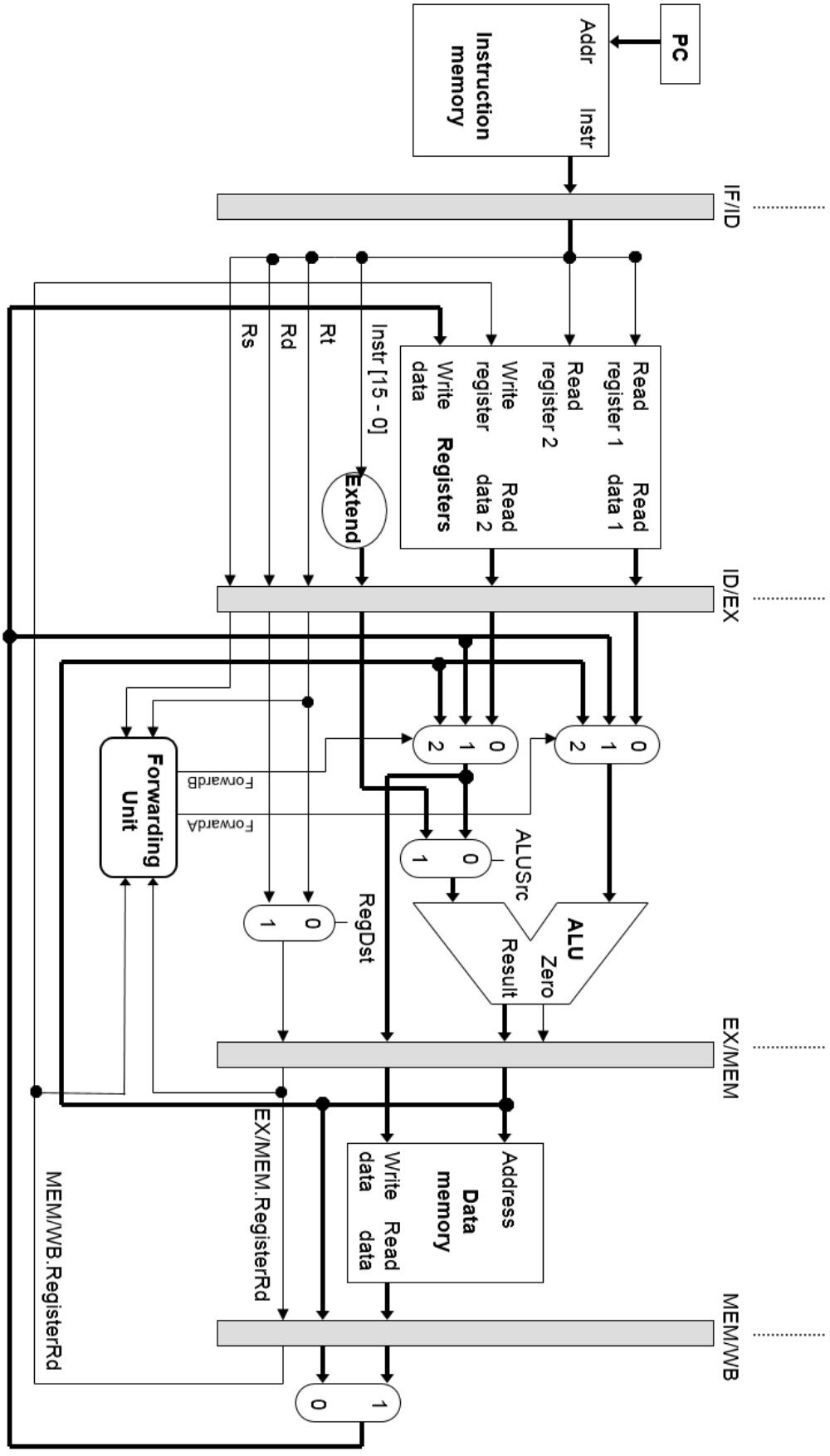
תרשים מספר 1


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

תרשים מספר 2


start
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
1
ALUSrcA = 00
ALUSrcB = 11
ALUOp = 00
ReadSrc = 0
LW or SW
R-type
BEQ
J
ADD3
2
ALUSrcA = 01
ALUSrcB = 10
ALUOp = 00
LW
SW
3
MemRead
IorD = 1
4
RegDst = 0
RegWrite
MemtoReg = 1
5
MemWrite
IorD = 1
6
ALUSrcA = 01
ALUSrcB = 00
ALUOp = 10
7
RegDst = 1
RegWrite
MemtoReg = 0
8
ALUSrcA = 01
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
10
ALUSrcA = 01
ALUSrcB = 00
ALUOp = 00
ReadSrc = 1
11
ALUSrcA = 10
ALUSrcB = 00
ALUOp = 00

תרשים מספר 3


PC
Addr
Instr
Instruction memory
IF/ID
Rs
Rd
Rt
Instr [15 - 0]
Extend
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
ID/EX
Forward B
Forward A
Forwarding Unit
ALUSrc
RegDst
ALU
Zero
Result
EX/MEM
EX/MEM.RegisterRd
Address
Data memory
Write data
Read data
MEM/WB
MEM/WB.RegisterRd