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 א'

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> <u>(45 נקודות):</u>

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

נתון רצף פקודות האסמבלי הבא:

add	$5	$1	$3
sub	$2	$1	$3
and	$8	$2	$5
or	$8	$5	$2
lw	$2	8($8)	
and	$1	$2	$2
sw	$4	0($1)	
sub	$1	$2	$3
sw	$1	0($4)	

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

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

א. כיצד יש לתזמן את הרכיבים השונים במעבד ה- Pipelined MIPS על מנת שלא תיווצר בעיה בביצוע הפקודות add ו- or כפי שהן מופיעות ברצף הנ"ל? הסבירו היטב!

יש לתזמן את ה- Register File כך שידגום בירידת השעון (negative-edge triggered), ואת כל שאר הרכיבים המתוזמנים יש לתזמן כך שידגמו בעליית השעון (positive-edge triggered). באופן זה, פעולת ה- WB של פקודת ה- add תתבצע בחצי הראשון של מחזור השעון החמישי (משום שהדגימה אל תוך ה- Register File תתבצע בירידת השעון), והקריאה של הערך העדכני תתבצע ע"י פקודת ה- or במחצית השנייה של מחזור השעון החמישי (בשלב ה- ID, כאשר הדגימה אל ה- pipeline-register תתבצע בעליית השעון של סוף מחזור השעון החמישי – תחילת מחזור השעון השישי). אם כל הרכיבים היו מתוזמנים כך שידגמו בעליית שעון (positive-edge triggered), הרי שהפקודה or לא הייתה קוראת את הערך המעודכן של $5, משום שהוא היה מתעדכן רק **בסוף** זמן המחזור החמישי, עם עליית השעון.

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

# Clock cycle / Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
add $5 $1 $3	IF	ID	EX	M	WB										
sub $2 $1 $3		IF	ID	EX	M	WB									
and $8 $2 $5			IF	ID	EX	M	WB								
or $8 $5 $2				IF	ID	EX	M	WB							
lw $2 8($8)					IF	ID	EX	M	WB						
and $1 $2 $2						IF	ID	--	EX	M	WB				
sw $4 0($1)							IF	--	ID	EX	M	WB			
sub $1 $2 $3								--	IF	ID	EX	M	WB		
sw $1 0($4)										IF	ID	EX	M	WB	

- היעזרו בתרשים מספר 1 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	0	0	0	1
4	0	0	0	1
5	2	1	0	1
6	0	1	0	1
7	2	x	1	0
8	x	x	x	x
9	1	1	0	1
10	2	0	1	x
11	0	0	0	1
12	1	2	1	x
13				
14				
15				

ד. כמה מחזורי שעון היה לוקח לתכנית הנ"ל לרוץ על מעבד מצונר **שאיננו** תומך ב- forwarding?
מהו ה- speedup שהושג כתוצאה משימוש ביחידת ה- forwarding עבור התכנית הנתונה?

אם תכנית הייתה רצה על מעבד מצונר שאיננו תומך ב- forwarding אזי:

- אם ה- Register File היה מתוזמן כך שידגום בירידת השעון (negative-edge triggered), הרי שהיה עלינו לעכב את ביצוע פקודה מספר 3 בשני מחזורי שעון (כדי שיהיה אפשר לקרוא את $2 המעודכן), וכן לעכב את ביצוע פקודה מספר 5 בשני מחזורי שעון (כדי שיהיה אפשר לקרוא את $8 המעודכן). את ביצוע פקודה מספר 6 נאלץ לעכב בשני מחזורי שעון (כדי שנוכל לקרוא את $2 המעודכן), וכן את פקודות 7 ו- 9 נאלץ לעכב בשני מחזורי שעון כל אחת (בשביל שנוכל לקרוא את הערך העדכני של $1 בכל פעם). בסה"כ, ידרשו 23 מחזורי שעון כדי להריץ את התכנית על מעבד מצונר שאינו מכיל יחידת forwarding, וזאת בהשוואה ל- 14 מחזורי שעון שנדרשים להריץ את אותה תכנית על מעבד שכן תומך ב- forwarding.

  במקרה הזה: $Speedup = \frac{Time_{old}}{Time_{new}} = \frac{23}{14} \simeq 1.64$

- אם ה- Register File לא מתוזמן כך שידגום בירידת שעון, הרי שאין אפשרות לכתוב לרגיסטר מסוים ולקרוא את הערך המעודכן מאותו רגיסטר באותו מחזור שעון, ולכן, את כל אחת מהפקודות הנ"ל נאלץ לעכב בשלושה מחזורי שעון, ולא בשניים. ולכן, ידרשו 28 מחזורי שעון לשם הרצת התכנית על מעבד מצונר שאינו מכיל יחידת forwarding, וזאת בהשוואה ל- 14 מחזורי שעון שנדרשים להריץ את אותה תכנית על מעבד שכן תומך ב- forwarding.

  במקרה הזה: $Speedup = \frac{Time_{old}}{Time_{new}} = \frac{28}{14} \simeq 2$

<u>שאלה מספר 2</u> (45 נקודות):

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

הפקודה Load Immediate טוענת ערך immediate אל תוך רגיסטר ב- register file. דהיינו:

```
li  $s0, imm          #  $s0 ← imm
```

א. עליכם לבצע הוספת חומרה ו/או שינויים במסלול הנתונים המתואר בתרשים מספר 2 כך שהחומרה תתמוך בביצוע הפקודה **תוך 3 מחזורי שעון**. אין לשנות את ה- ALU. יש להסביר במחברת באופן מילולי ומפורט את כל השינויים שנעשו על גבי הסכמה.

ראו תרשים מספר 2. שימו לב כי אות הבקרה MemtoReg הינו כעת בעל שתי סיביות.

ב. שנו את מכונת המצבים של הבקר בתרשים מספר 3 כך שהפקודה החדשה תתבצע **תוך 3 מחזורי שעון**. יש להעדיף שימוש במצבים קיימים. אם אתם מוסיפים או משנים מצבים יש לרשום בתוכם את ערכי אותות הבקרה וה- enables הרלוונטיים.

ראו תרשים מספר 3. שימו לב כי יש צורך לעדכן את MemtoReg גם במצבים 4 ו- 7.

ג. כעת, אתם נדרשים לבצע הוספת חומרה ו/או שינויים במסלול הנתונים של ה- Pipelined MIPS המתואר בתרשים מספר 4, על מנת לממש את פקודת ה- li כך שהנתונים יעברו מסלול הדומה למה שעשיתם בסעיף א' (בשינויים המחויבים לארכיטקטורת ה- pipeline, כמובן).

ראו תרשים מספר 4.

ד. בוגר מצטיין של הפקולטה להנדסה בבר-אילן הצליח לממש את הפקודה li בארכיטקטורת Pipelined MIPS **מבלי** לשנות או להוסיף שום דבר במסלול הנתונים. הסבירו היטב כיצד ניתן לעשות זאת, ואיזה מחיר יש לכך.

ניתן לעשות זאת ע"י שימוש בחומרה הקיימת בלבד, תוך ניצול של השדה ה- rs הפנוי בקידוד הפקודה. הרעיון הוא להגדיר את הפקודה כך ששדה ה- rs יצביע לכתובת של $zero (חמישה אפסים), ואז לבצע סכימה של הערך ה- immediate (אחרי שעבר sign extension) ביחד עם הערך האגור ב- $rs (שם אגור ערך 0). למעשה, המימוש דומה לפקודת ה- addi עליה למדנו בתרגול, ורעיון הוא לנצל את שדה ה- rs כדי לממש:

$rt = $rs + imm = $zero + Imm = Imm

אפשר וראוי להשתמש באותו opcode של פקודת ה- addi, וכל מה שנותר לעשות הוא להגדיר את שדה ה- rs כך שיכיל חמישה אפסים כחלק מהגדרת הפורמט של הפקודה:

ADDI	rs	rt	immediate
0 0 1 0 0 0	0 0 0 0 0		

ה. הראו כיצד ניתן לממש את הפקודה li גם בארכיטקטורת Multi-Cycle MIPS מבלי לשנות או להוסיף שום דבר במסלול הנתונים. הסבירו היטב כיצד ניתן לעשות זאת, ואיזה מחיר יש לכך. עדכנו בהתאם את מכונת המצבים של הבקר על גבי תרשים מספר 5.

הרעיון זהה למה שמוסבר בסעיף הקודם, רק שהפעם יש לכך מחיר, שהרי לא ניתן יהיה לממש את הפקודה תוך 3 מחזורי שעון (כפי שעשינו בסעיפים א' + ב'), אלא כעת נדרשים 4 מחזורי שעון. מחיר נוסף שיש לכך הוא מחיר אנרגטי, שנובע מפעולת ה- ALU הנוספת. למעשה, עדכון מכונת המצבים של הבקר בתרשים מספר 5 זהה למה שלמדנו בתרגול לגבי מימוש פקודת addi. ראו תרשים מספר 5.

ו. מה יהיה זמן הביצוע (execution time) של תכנית בעלת שלושה מיליון פקודות הרצה על מעבד מסוג Multi-Cycle MIPS העובד בתדר של 100MHz? התייחסו בתשובתכם הן לארכיטקטורה מסעיף א' והן לארכיטקטורה מסעיף ה'. חלוקת הפקודות בתכנית נתונה בטבלה הבאה:

סוג הפקודה	אחוזים מסך הפקודות
Load	20%
Store	12%
Branch	18%
Jump	2%
R-Type	33%
Li	15%

זמן הביצוע נתון ע"י הנוסחה: $Execution\ time = IC \cdot CPI_{eff} \cdot CCT$,

כאשר תדר השעון נתון, וכן נתון מספר הפקודות בתכנית. כל שנותר לחשב הוא את ה- CPI_{eff}.

עבור הארכיטקטורה מסעיף א' נקבל כי:

סוג הפקודה	אחוזים מסך הפקודות	מספר מחזורי השעון
Load	20%	5
Store	12%	4
Branch	18%	3
Jump	2%	3
R-Type	33%	4
Li	15%	3

ולכן, נקבל כי:

$$CPI_{eff} = 5 \cdot 20\% + 4 \cdot 12\% + 3 \cdot 18\% + 3 \cdot 2\% + 4 \cdot 33\% + 3 \cdot 15\% = 3.85$$

ולכן:

$$Execution\ time = IC \cdot CPI_{eff} \cdot CCT = 3 \times 10^6 \cdot 3.85 \cdot \frac{1}{100 \times 10^6} = 115.5[\text{mSec}]$$

עבור הארכיטקטורה מסעיף ה' נדרשים 4 מחזורי שעון לביצוע פקודת Li, ולכן נקבל כי:

$$CPI_{eff} = 5 \cdot 20\% + 4 \cdot 12\% + 3 \cdot 18\% + 3 \cdot 2\% + 4 \cdot 33\% + 4 \cdot 15\% = 4$$

ולכן: $$Execution\ time = IC \cdot CPI_{eff} \cdot CCT = 3 \times 10^6 \cdot 4 \cdot \frac{1}{100 \times 10^6} = 120[\text{mSec}]$$

שאלה מספר 3 (20 נקודות):

בשאלה זו נעסוק בתכנון וניתוח ביצועים של מערכת זיכרון.

- נתון מעבד עם מערכת זיכרון בעלת רמה אחת של זיכרון מטמון (L1). נתון כי זמן הגישה הממוצע לזיכרון הוא 2.4 מחזורי שעון, כאשר:
  - אם המידע נמצא בזיכרון המטמון אזי זמן הגישה לזיכרון הוא מחזור שעון יחיד,
  - אם המידע לא נמצא בזיכרון המטמון אזי נדרשים עוד 80 מחזורי שעון על מנת להביא את המידע מהזיכרון הראשי (main memory).

צוות מהנדסים מנסה לשפר את זמן הגישה הממוצע לזיכרון, במטרה להשיג שיפור של 65% בזמן הגישה הממוצע (speedup=1.65). צוות המהנדסים החליט להוסיף רמה נוספת של זיכרון מטמון (L2), כאשר נתון כי:

- זמן הגישה ל- L2 הינו 6 מחזורי שעון.
- הוספת ההיררכיה של L2 למערכת הזיכרון איננה משפיעה כלל על רצף הגישות ל- L1 או על זמני הגישות ל- L1.
- זמן הגישה לזיכרון הראשי נותר כשהיה (80 מחזורי שעון).

א. מה צריך להיות ה- hit rate של L2 על מנת להשיג את השיפור הרצוי ב- AMAT?

ראשית, עלינו למצוא מהו ה- miss rate של L1:

$$AMAT = HitTime + MissRate_{L1} \times MissPenalty$$

$$2.4 = 1 + MissRate_{L1} \times 80$$

$$MissRate_{L1} = 1.75\%$$

בשלב הבא, נרצה לחשב מהו ה- AMAT שעלינו לחתור אליו על מנת להשיג את השיפור הנדרש בשאלה:

$$Speedup = \frac{Time_{old}}{Time_{new}}$$

$$1.65 = \frac{2.4}{Time_{new}}$$

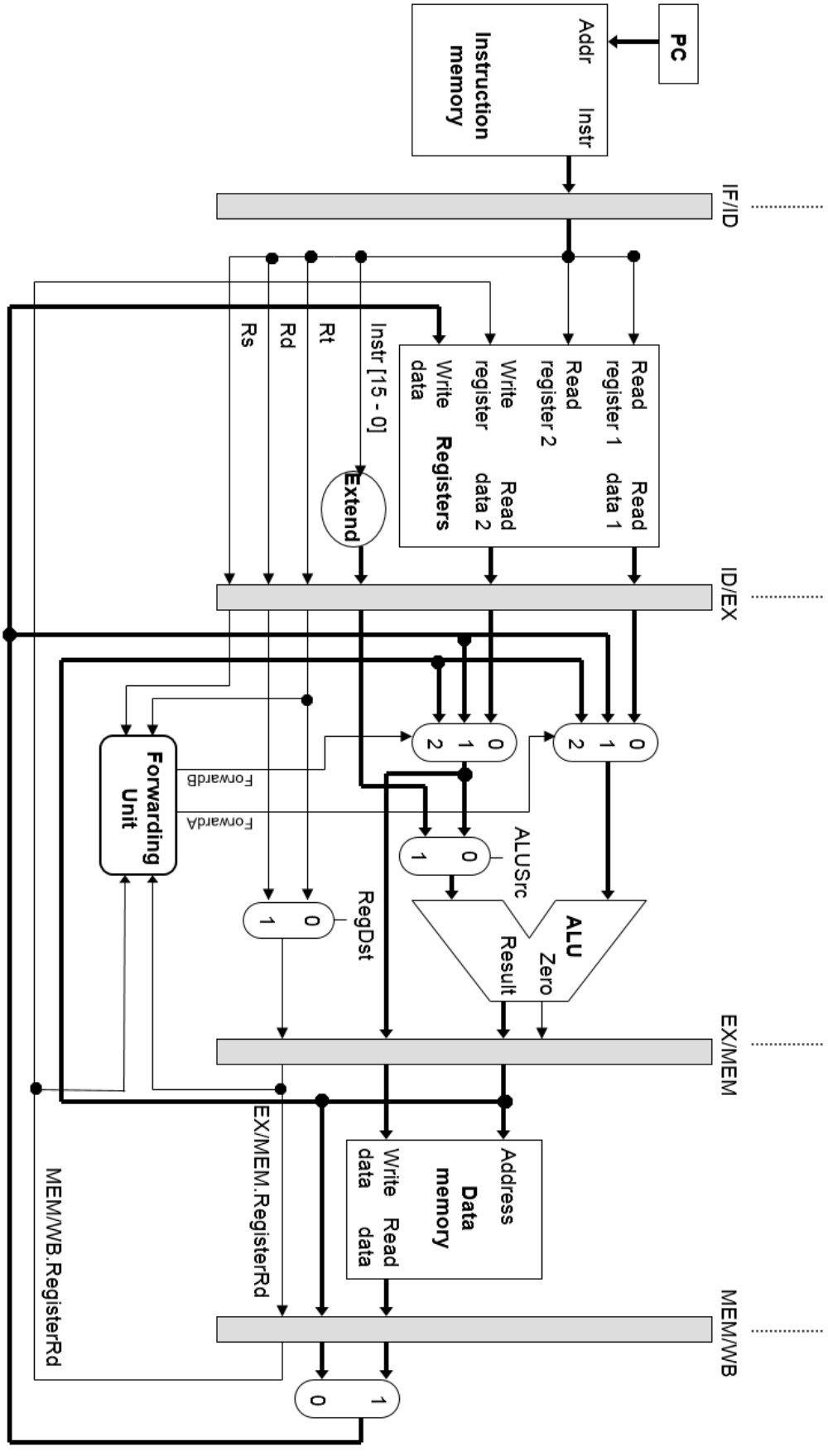
$$Time_{new} = 1.4545 \ [\text{clock cycles}]$$

כעת, נוכל להשתמש שוב בנוסחת ה- AMAT על מנת למצוא את ה- miss rate של L2 שיניב את השיפור הרצוי:

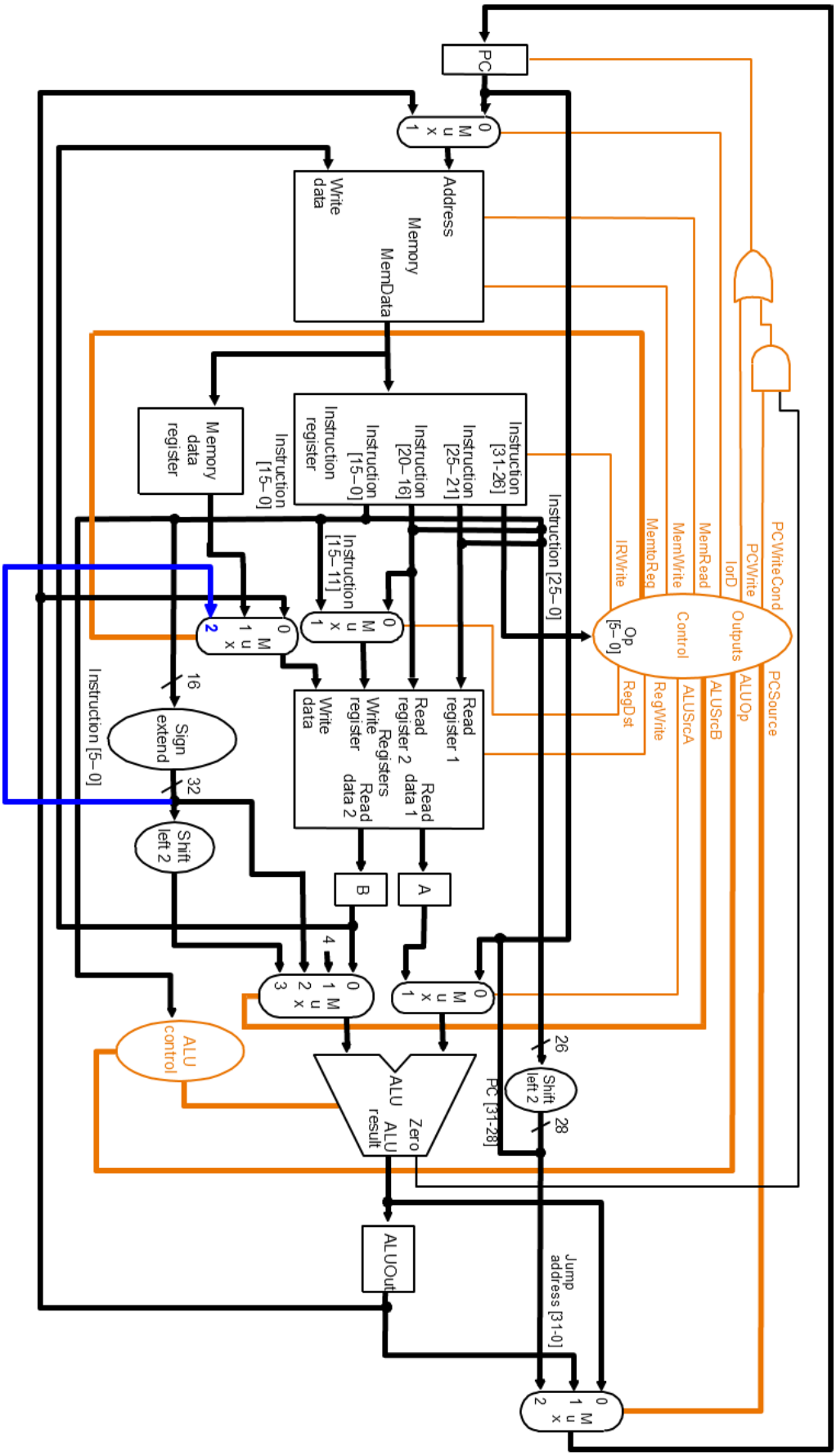
$$1.4545 = 1 + 0.0175 \times \left(6 + MissRate_{L2} \times 80\right)$$

מפתרון של המשוואה הנ"ל עולה כי ה- miss rate הגבוהה ביותר האפשרי עבור L2 הוא: ≈ 0.2497, ולכן, ה- hit rate של L2 צריך להיות לפחות: $\sim 75\%$

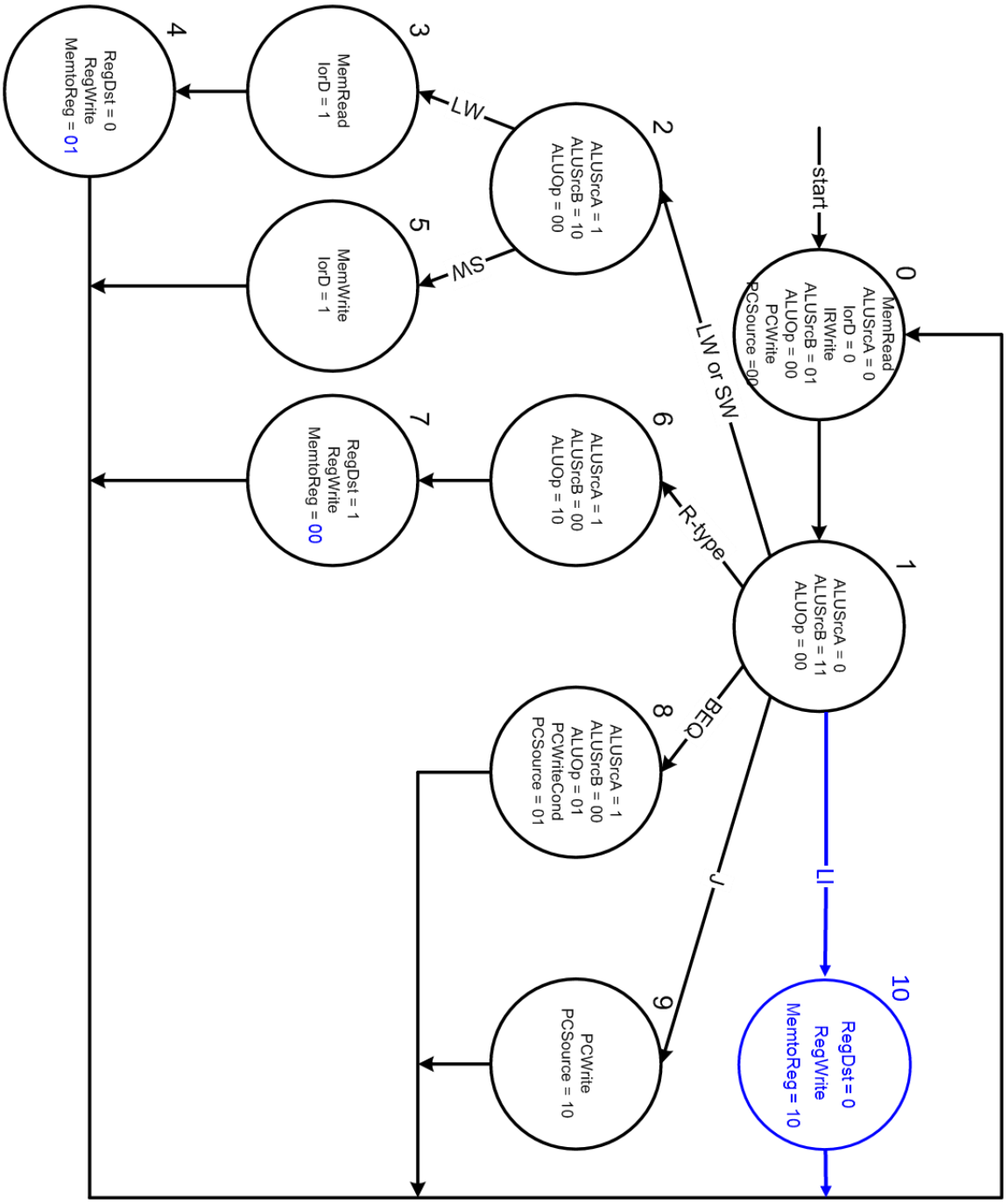
תרשים מספר 1


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

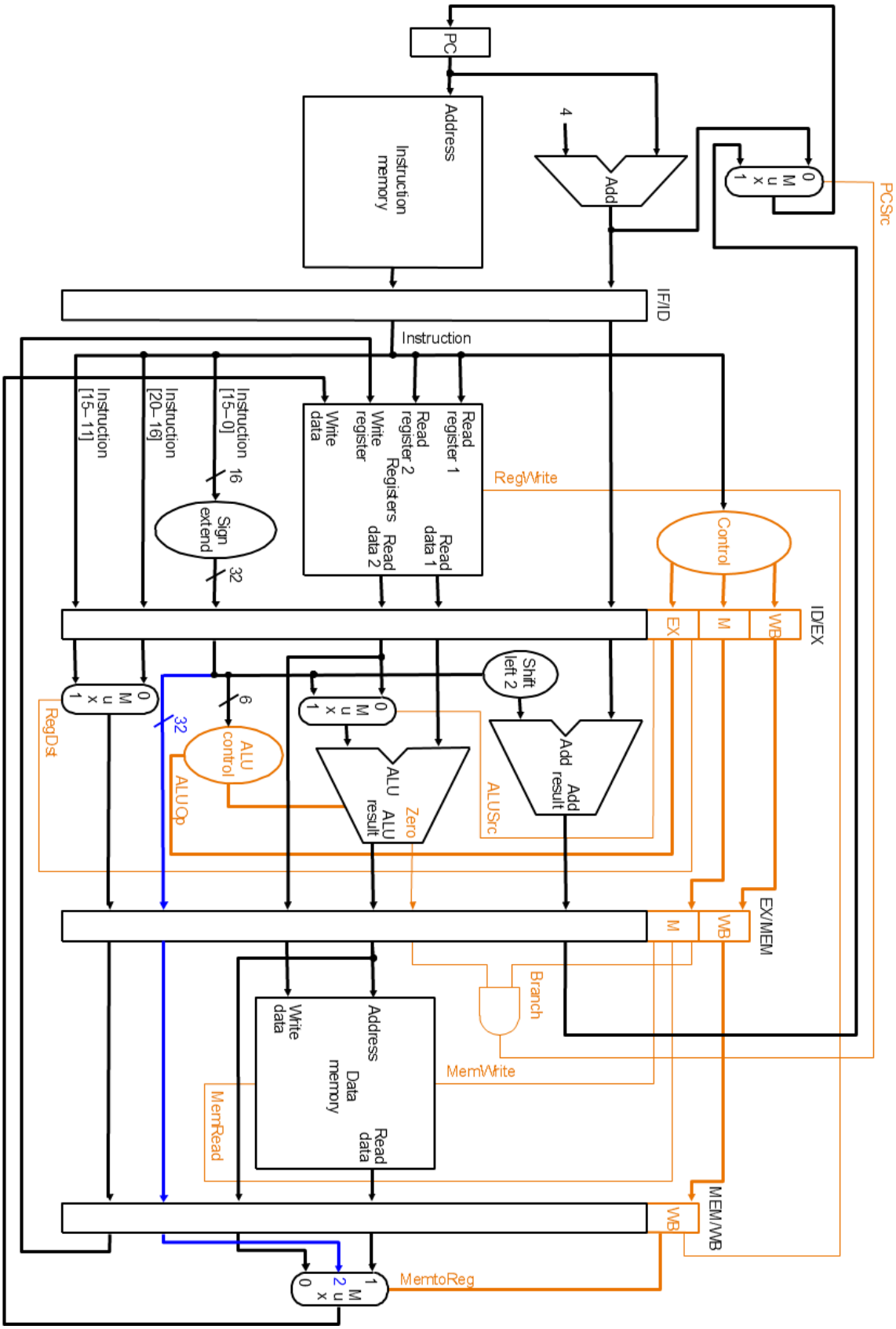
תרשים מספר 2


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

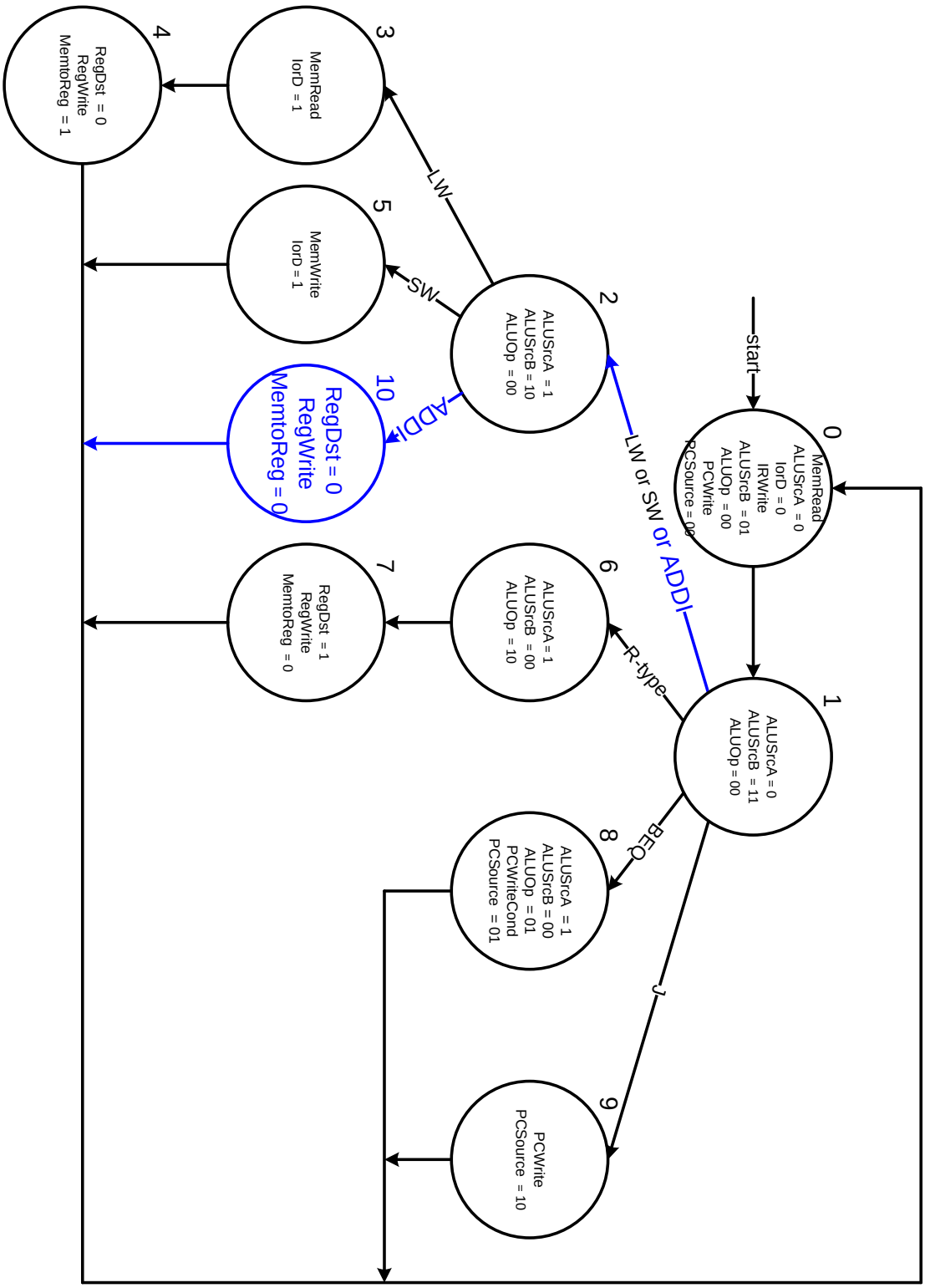
תרשים מספר 3


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

תרשים מספר 4


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 2
Registers
Read data 2
Read register 1
Read data 1
RegWrite
Control
EX
M
WB
ID/EX
Shift left 2
RegDst
6
32
ALU control
ALUOp
ALU
ALU result
Zero
ALUSrc
Add
Add result
EX/MEM
Branch
MemWrite
MemRead
Write data
Address
Data memory
Read data
MEM/WB
MemtoReg

תרשים מספר 5


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