מבנה מחשבים

236267

תרגיל בית יבש 4

מגישים:

יואב יעבץ	212617864	yoavjavits@campus.technion.ac.il
גלעד שמרלר	212139240	shmerler@campus.technion.ac.il

שאלה 1

נתון קטע הקוד הבא:

```
div R7, R6, R5      \R7 = R6/R5
mul R8, R7, R7      \R8 = R7*R7
sub R6, R3, R7      \R6 = R3-R7
bne Label           \jump if last result not 0
add R5, R9, R1      \R5 = R9+R1
Label:
store [R5 + 0X4], R2 \Mem[R5+4]=R2
Load R1, [R3 + 0x11] \R1=Mem[R3+17]
```

<u>להלן הנתונים הבאים</u>:

- ב - ROB וב – ROB Cont 10 כניסות.
- בתחילת ביצוע קטע הקוד, ה - ROB וה – ROB Cont ריקים.
- בשלב ה - ALLOC 3 פקודות לכל היותר מתבצעות בכל זמן מחזור, השלב לוקח מחזור שעון אחד עבור כל פקודה, ורק החל מהמחזור הבא ניתן להתחיל לבצע exe.
- בשלב ה - Commit 3 פקודות לכל היותר מתבצעות בכל זמן מחזור.
- במעבד קיימות 4 היחידות הפונקציונליות הבאות: IEU, JEU, IMUL, IDIV

עבור הרכיבים הבאים נתון:

רכיב	משך הביצוע (במחזורי שעון)
IEU – Integer Execution Unit	1
JEU – Jump Execution Unit	1
IMUL – Integer Multiply	10
IDIV – Integer Divide	40

- החיזוי לגבי פקודות הסתעפות הוא תמיד Not Taken, כאשר במקרה של Misprediction, ה – Flush מתבצע בסיום שלב ה – EXE.

- תוכן ה - RRF לפני ביצוע הקוד הוא (הערכים השונים נתונים בייצוג עשרוני):

RRF:

R1	3
R2	4
R3	8
R4	1
R5	2
R6	10
R7	8
R8	6
R9	18

<u>הערה</u>: לאורך כל הסעיפים, ניתן לצורך נוחיותכם לכתוב בכל טבלה רק את השינויים ביחס לסעיף הקודם.

1) ציינו את תוכן הכניסות ב – RAT, RRF, ROB, ROB Cont בסיום שלב ה - ALLOC של הפקודה האחרונה:

RAT:

	#reg	ROB/RRF
R1	P7	ROB
R2		RRF
R3		RRF
R4		RRF
R5	P5	ROB
R6	P3	ROB
R7	P1	ROB
R8	P2	ROB
R9		RRF

RRF:

R1	3
R2	4
R3	8
R4	1
R5	2
R6	10
R7	8
R8	6
R9	18

ROB:

Entry #	Valid Entry	Valid Data	Data	Destination
1	1	0		R7
2	1	0		R8
3	1	0		R6
4	1	0		PC
5	1	1	21	R5
6	1	0		MEM
7	1	0		R1
8	0			
9	0			
10	0			

ROB Cont:

Valid Entry	opcode	Valid SRC1	SRC1	Valid SRC2	SRC2	Destination
1	Div	1	10	1	2	P1
1	Mul	0	P1	0	P1	P2
1	Sub	1	8	0	P1	P3
1	Bne	0	P3			P4
1	Add	1	18	1	3	P5
1	STA	1	21	1	4	Buffer
1	STD	1	4			Buffer
1	Load	1	8	1	17	P7
0						
0						

2) ציינו את תוכן הכניסות ב – RAT, RRF, ROB, ROB Cont בסיום שלב ה - EXE של פקודת ההסתעפות בשורה 4 - *bne Label*.
שימו לב כי מדובר כאן על **לפני** שלב הflush.

RAT:

	#reg	ROB/RRF
R1	P7	ROB
R2		RRF
R3		RRF
R4		RRF
R5	P5	ROB
R6	P3	ROB
R7		RRF
R8	P2	ROB
R9		RRF

RRF:

R1	3
R2	4
R3	8
R4	1
R5	2
R6	10
R7	5
R8	6
R9	18

ROB:

Entry #	Valid Entry	Valid Data	Data	Destination
1	0			
2	1	0		R8
3	1	1	3	R6
4	1	1	Taken	PC
5	1	1	21	R5
6	1	1	4	MEM
7	1	1	Mem[25]	R1
8	0			
9	0			
10	0			

ROB Cont:

Valid Entry	opcode	Valid SRC1	SRC1	Valid SRC2	SRC2	Destination
0						
1	Mul	1	5	1	5	P2
0						
0						
0						
0						
0						
0						
0						
0						

3) ציינו את תוכן הכניסות ב – RAT, RRF, ROB, ROB Cont בסיום שלב ה - EXE של פקודת הstore כאשר אנו מבצעים אותה עם הערכים הנכונים של ריצת התוכנית (רמז – האם יש רק ביצוע אחד של אותה פקודת הstore?):

RAT:

	#reg	ROB/RRF
R1	P3	ROB
R2		RRF
R3		RRF
R4		RRF
R5		RRF
R6		RRF
R7		RRF
R8		RRF
R9		RRF

RRF:

R1	3
R2	4
R3	8
R4	1
R5	2
R6	3
R7	5
R8	25
R9	18

ROB:

Entry #	Valid Entry	Valid Data	Data	Destination
1	1	1	2	R5
2	1	0		MEM
3	1	0		R1
4				
5				
6				
7				
8				
9				
10				

ROB Cont:

Valid Entry	opcode	Valid SRC1	SRC1	Valid SRC2	SRC2	Destination
1	STA	1	2	1	4	Buffer
1	STD	1	4			Buffer
1	Load	1	8	1	17	P3

4) ציינו את תוכן הכניסות ב – RAT, RRF, ROB, ROB Cont בסיום שלב ה - Commit של פקודת הload :

RAT:

	#reg	ROB/RRF
R1		RRF
R2		RRF
R3		RRF
R4		RRF
R5		RRF
R6		RRF
R7		RRF
R8		RRF
R9		RRF

RRF:

R1	Mem[25]
R2	4
R3	8
R4	1
R5	2
R6	3
R7	5
R8	25
R9	18

ROB:

Entry #	Valid Entry	Valid Data	Data	Destination
1	0			
2	0			
3	0			
4	0			
5	0			
6	0			
7	0			
8	0			
9	0			
10	0			

ROB Cont:

Valid Entry	opcode	Valid SRC1	SRC1	Valid SRC2	SRC2	Destination
0						
0						
0						
0						
0						
0						
0						
0						
0						
0						

שאלה 2

א. יש למלא את הטבלה שבהמשך. לכל פקודה יש לרשום:

- R3, R2, R1 – ערכי הרגיסטרים הארכיטקטוניים לאחר commit של הפקודה.
- addr – כתובת הגישה לזיכרון – עבור פקודות load ו-store בלבד.
- data – ערך זיכרון שנקרא או נכתב – עבור פקודות load ו-store בלבד.

כבר מולאו

- T alloc: הזמן בו מבוצעת אלוקציה: **עד 3 פקודות בכל מחזור**, החל מ- t =1.
ניתן לבצע אלוקציה רק כאשר **<u>לכל הפקודות</u>** שמספק ה-frontend במחזור יש מקום ב-Cont ROB וב-ROB.
  - ב-ROB ישנן **10 כניסות וב-ROB Cont 6**. כל פקודה (כולל פקודות Store) תופסת מקום אחד ב-ROB Cont.
- src1, src2: מספרי הרגיסטרים המשמשים כ-sources לפקודה: Pi עבור רגיסטר פיזי, ו-Ri במידה וקוראים ישירות את הרגיסטר הארכיטקטוני.
  - עבור store: src1 – הרגיסטר המשמש לחישוב הכתובת. src2 – הרגיסטר המכיל את הנתון.
  - עבור פקודת jump if zero(JZ): פקודת קפיצה מותנית: בצע קפיצה אם Zero flag=1:
    - פקודת Add, Sub(ALU) שאם תוצאת החישוב שלה היא 0 מעדכנת את הדגל ל-1 ואחרת ל-0.
    - הדגל מהווה dst נוסף של פקודות ALU, ומשמש כ-src עבור פקודת JZ.
    - הדגל עובר renaming, בדומה לרגיסטרים: PZi אם נכתב ע"י פקודה i, ו-Z אם ארכיטקטוני.
- T src1 ready , T src2 ready: הזמן בו מוכן כל אחד ערכי ה-sources לפקודה.
אם ה-src כבר מוכן בזמן האלוקציה, אז זמן זה יהיה שווה לזמן האלוקציה.
אחרת, זמן זה שווה ל-T data ready של הפקודה שמחשבת את הערך של ה-src.
- T exe: הזמן בו הפקודה נשלחת לביצוע. הניחו כי ישנן אינסוף יחידות ביצוע.
  - פקודה יכולה להיכנס לביצוע לכל המוקדם במחזור שלאחר האלוקציה.
  - פקודה נכנסת לביצוע במחזור השעון שלאחר המחזור בו כל ה-src-ים מוכנים. פקודת store נכנסת לביצוע במחזור השעון שלאחר המחזור בו src1 (המשמש לחישוב הכתובת) מוכן, וכאשר קיימת יחידת ביצוע מתאימה פנויה.
  - במעבד קיימות יחידות הביצוע הבאות:
  - יחידת sub/add עבור ביצוע פעולות חיבור או חיסור. משך הביצוע: מחזור 1
  - יחידת AGU עבור ביצוע פעולות load/store. משך הביצוע: מחזור 1
  - יחידת jump עבור ביצוע פעולות jump. משך הביצוע: מחזור 1
  - יחידת div עבור ביצוע פעולות div. משך הביצוע: 5 מחזורים
היחידה היא pipelined, כך שבכל שני מחזורים ניתן להתחיל פקודה חדשה.
  - פקודת מוצאת מה-ROB Cont במחזור שלאחר סיום הביצוע שלה, וכבר במחזור זה פקודה חדשה יכולה לבצע alloc ולהשתמש במקום שהתפנה.
- Load block code: עבור load שנשלח לביצוע בזמן t=Texe, או שהוסר עבורו תנאי חסימה קודם בזמן t, תנאי החסימה נבדקים בזמן t+1 לפי הסדר (רשמו את כל תנאי החסימה לפי הסדר):
  - 1 – חסימה כתוצאה מ- unknown store address
  - 2 – חסימה כתוצאה מ- waiting for store data

- T addr ready : ממולא עבור פקודות load ו-store בלבד : **Texe+1**

עבור load ו-store המבוצעים באותו זמן t : תנאי החסימה של ה-load נבדקים בזמן t+1, בזמן זה הכתובת של ה-store ידועה, ולכן ה-load לא נחסם על unknown store address ע"י ה-store.

- T data ready :
  - עבור פקודות ALU : פקודות Add,Sub : **Texe+1**, פקודת Div : **Texe+5**.
  - עבור load שהוסרו עבורו כל תנאי החסימה בזמן t או שבוצע בזמן t ולא נחסם :
    - במידה וה-load פוגע ב-cache, או שיש store to load forwarding : בזמן **t+4**.
    - אחרת, במידה ובוצע load אחר לאותה שורה ב-cache בזמן t'<t : בזמן **max(t'+10, t+4)**.
    - אחרת, בזמן **t+10**.
  - עבור store : מחזור השעון בו הן ה data-לכתיבה לזיכרון והן הכתובת מוכנים.
  - כלומר T data ready = max(Texe+1, T src2 ready).
  - הכתובת של ה store-ידועה בזמן t=texe+1.
  - בזמן זה מוסר תנאי החסימה של load שנחסם ע"י הstore על ה address store unknown.
  - בפרט load, שמבוצע בזמן t, לא נחסם על unknown store address על ידי הstore, אך load שמבוצע בזמן t> נחסם.

- ROB Cont entries - מספר הכניסות התפוסות ב ROB Cont -לאחר האלוקציה של הפקודה הנתונה (בהתחשב גם בפקודות שהוצאו מה ROB Cont -במחזור זה.)
- ROB entries - מספר הכניסות התפוסות ב ROB -לאחר האלוקציה של הפקודה הנתונה (בהתחשב גם בפקודות שהוצאו מה ROB -במחזור זה.)

- עבור פקודת Jump עם חיזוי שגוי, מבוצע flush בזמן Texe+1, והפקודות מהמסלול הנכון יכולות לבצע אלוקציה החל מזמן **Texe+6** (במידה ואין סיבה אחרת שמעכבת את האלוקציה).

- T commit : הזמן בו הפקודה מבצעת commit. ניתן לבצע commit לעד **3 פקודות** בכל מחזור.
  - פקודה יכולה לבצע commit החל מזמן T data ready+1.
    וכבר במחזור זה פקודה חדשה יכולה לבצע alloc ולהשתמש במקום שהתפנה.

- הנחות :
  - L1 data cache : גודל שורה $32_{10}B = 20_{16}B$. write no allocate. ריק בתחילת הביצוע.
  - הכתובות הן פיזיות (אין צורך בתרגום) וכל הערכים המספריים הם בבסיס 16.
  - הניחו כי ה frontend-יכול לספק 3 פקודות בכל מחזור.
  - בטבלה רשומות אך ורק הפקודות מהמסלול הנכון.

Pdst	instruction	R1	R2	R3	addr	data	src1	src2	T alloc	T src1 ready	T src2 ready	T exe	block code	T data ready	T commit	#RS entries	#ROB entries
0	Div R1←R2 / 2	40	80	160			R2		1	1		2		7	8	3	3
1	store m[R1+10] ← R3	40	80	160	50	160	P0	R3	1	7	1	8		9	10	3	3
2	load R2 ← m[R2+10]	40	60	160	90	60	R2		1	1		2	1	19	20	3	3
3	Div R1←R3 / 2	80	60	160			R3		2	2		4		9	20	6	6
4	store m[R1-10] ←R2	80	60	160	70	60	P3	P2	2	9	19	10		19	20	6	6
5	load R1 ←m[R3-90]	60	60	160	70	60	R3		2	2		3	1,2	23	24	6	6
6	sub R3 ←R3 – 20	60	60	140			R3		8	8		9		10	24	6	7
7	JZ 1000 wrongly predicted	60	60	140			PZ6		10	10		11		12	24	6	7
8	add R3 ←R3 – 20	60	60	120			P8		20	24		25		26	27	4	7

שאלה 3

לאחר אקזיט מוצלח, שבוזבז כולו על כובע #1 בביקיני בוטום, ניסים מזרחי אדרי בן זכאי חיפש דרך חדשה לעשות כסף. הוא הפעיל את קשריו בחברת המעבדים הגדולה, לטניא, והציע להם להוסיף פקודות חדשות לארכיטקטורת RISCV החדשה אותה הם מפתחים תוך שימוש בOOOE. הוא חשב שהפקודות הבאות יהיו שימושיות:

1. **Load-Jump**

$$LJ\ R1, R2, R3, label$$

הפקודה תפעל בצורה הבאה:

1. טעינת הערך מהזיכרון מהכתובת $Mem[R2]$
2. השוואה בין $Mem[R2]$ ל-$R3$

א. במקרה של ערך $R1 = Mem[R2]\ True$ קופצים לכתובת label.

ב. במקרה של ערך False:

$R1$ לא משתנה, לא קופצים.

2. **Store-Jump**

$$SJ\ R1, R2, R3, label$$

הפקודה תפעל בצורה הבאה:

1. השוואה בין $R2, R3$

א. במקרה של ערך True, אז $Mem[R1] = R2$ וקופצים לכתובת label.

ב. במקרה של ערך FALSE: אין כתיבה לזיכרון, לא קופצים.

הכתובת והDATA של פקודת הStore תמיד מסיימות את החישוב שלהן גם אם בסוף מגלים שערך ההשוואה הוא FALSE, במצב זה לא תתרחש כתיבה לזיכרון.

שאלות:

א.

ROB				
#	Valid	Dest	Data v	Data
P0	1	Mem	1	
P1	1	R2	1	5
P2	1	PC	1	Taken

שורת ROB של פקודת Store

שורת ROB של פקודת Load

שורת ROB של פקודת Branch

בהנחה כי אפשר לכתוב לPC במקביל לכל רגיסטר אחר\לזיכרון על ידי הDEST הבא:

R1\MEM$$PC (במקרה זה הDATA יראה כך: Value$$Taken\Not Taken)

תנו דוגמה לשורת ROB אפשרית של פקודות LJ וS.

Command	#	Valid	Dest	Data V	Data
LJ	P0	1	R1$$PC	1	Mem[20]$$Taken
SJ	P1	1	Mem[15]$$PC	1	5$$Taken

ב. עבור פקודות LJ ו-SJ, כמה שורות Execution (ב"ROB Cont") נצטרך עבור כל פקודה? לכל אחת מהפקודות הציגו דוגמה למבנה "ROB Cont" לאחר הDecode של הפקודה.

<u>תשובה:</u>

עבור פקודת LJ – נצטרך שלוש פקודות, נפרט מדוע:

1. פקודה ראשונה – גישה לזיכרון בכתובת של הרגיסטר R2.
2. פקודה שנייה – השוואה בין ערך בכתובת הזיכרון מפקודה 1 לבין ערך הרגיסטר R3.
3. פקודה שלישית – עדכון ערך הרגיסטר PC במידת הצורך לפי ההשוואה הנעשתה בשלב הקודם; נשים לב כי במקביל נשנה את רגיסטר R1, ולפי הגדרת השאלה – ניתן לעשות זאת.

(המשך בעמוד הבא)

דוגמה לטבלת ROB Cont לאחר ה-Decode של הפקודה:

Valid Entry	Opcode	Valid src1	Src1	Valid src2	Src2	Dest
1	LJ – Load	1	10			P0
1	LJ – Compare	0	P0	1	13	P1
1	LJ - Jump	0	P1	1	P0$$Label	P2$$P3

כאשר מתקיים כי:

- P0 ממופה לערך Mem[R2].
- P1 ממופה לתוצאת ההשוואה בין Mem[R2] לבין R3.
- P2 ממופה לרגיסטר R1.
- P3 ממופה לרגיסטר PC.

עבור פקודת SJ – נצטרך שתי פקודות, נפרט מדוע:

1. פקודה ראשונה – השוואה בין רגיסטר R2 לבין רגיסטר R3.
2. פקודה שנייה – במקרה שהשוואה מהפקודה הראשונה היא אמת נעדכן את הזיכרון בכתובת ערך רגיסטר R1 להיות ערך רגיסטר R2, ונעדכן את הPC להיות זה של הכתובת Label. נשים לב כי ניתן לעשות את שני דברים אילו במקביל לפי הגדרת השאלה.
אחרת, במידה וההשוואה מהשלב הקודם אינו אמת לא נעשה כלום ונמשיך הלאה (נשים לב כי עדיין נחשב את החישובים הדרושים לפי הגדרת השאלה בשלב זה, אך זה כמובן לא יוסיף שלב נוסף).

דוגמה לטבלת ROB Cont לאחר ה-Decode של הפקודה:

Valid Entry	Opcode	Valid src1	Src1	Valid src2	Src2	Dest
1	SJ – Compare	1	10	1	15	P0
1	SJ – Save and Jump	0	P0	1	10$$label	P1$$P2

כאשר מתקיים כי:

- P0 מייצג את תוצאת ההשוואה הנערכת בשלב הראשון – בין הרגיסטרים R2 ל-R3.
- P1 מייצג את Mem[R1].

- P2 ממופה לרגיסטר PC.

ג. הסבירו כיצד שומרים על תקינות פקודת הLJ. כלומר, כיצד ניתן לוודא שערך הload יטען לרגיסטר אם״ם התנאי שוערך לTRUE (התייחסו בתשובתכם למה קורה אחרי סיום הבאת המידע מהזיכרון ולמה קורה ברגע שיערוך התנאי).

<u>תשובה:</u> נשים לב כי תכונה חשובה של המבנים בהם אנחנו משתמשים היא כי הפקודות מבצעות Commit לפי סדר כתיבתן. כל הפקודות המתבצעות לאחר שלב 2 שתיארנו לעיל תלויות בתוצאת ההשוואה שמתרחשת בו. לכן לפי התכונה שהזכרנו לעיל, כל פקודות המגיעות אחרי הפקודה הנ״ל לא יהיו יכולות לבצע Commit עד שפקודה זו ביצעה את ה-Commit שלה.
כעת נניח שאנחנו מבצעים עבור פקודה זו Commit. נפצל את התשובה שלנו לשתי אפשרויות:

המקרה הראשון, בו תוצאת ההשוואה היא אמת – נבצע Commit ואחריו נמשיך לעדכון הערכים המתאימים, כנדרש. המקרה השני, בו תוצאת ההשוואה היא שקר – כפי שלמדנו בכיתה, נבצע Flush ולכן לא נדאג משינוי ערך של הרגיסטרים – בדיוק כפי שנדרש. זאת מכיוון שלאורך הדרך אנחנו לא משנים את הרגיסטרים הארכיטקטוניים אלא רק את הרגיסטרים הפיזיים.

נתון קטע הקוד הבא:

```
1        SUB R2, R1, R3
2        SJE  R1, R2, R3, L1
3        ADD R1, R1, 4
4   L1:
5        DIV R1, R1, 1
6        LJNE R1, R1, R3, L2
7        SUB R3, R1, 4
8        LJE R1, R1, R2, L3
9   L2:
10       ADD R2, R3, 8
11  L3:
12
```

המצב ההתחלתי: R1=8, R2=0, R3=4

ד. מהן תוצאות ההרצה? (מה התוצאות הסופיות ברגיסטרים ובזיכרון שהשתנה)

<u>תשובה:</u> תוצאות ההרצה הן - $R1 = 4, R2 = 4, R3 = 4, Mem[8] = 4$.

ה.

לכל מחזור בתכנית (עד המחזור שאחרי הcommit האחרון כולל) רשמו אילו פקודות נמצאות במעבד (בטבלה) באופן דומה לזה שמולא בדוגמה למטה.

- מבנה הביצוע של כל פקודה: כאשר fetch, decode, commit לוקחים תמיד מחזור אחד, EXE משתנה בהתאם לפקודה.

•מוקצות ב־ROB 2 פקודות לכל היותר בכל מחזור.

•פקודה מתחילה להתבצע מחזור אחד לאחר שכל המקורות שלה מוכנים. בנוסף, פקודות load וstore צריכות מקום פנוי בload/store buffer בהתאם.

•ביצוע div לוקח 4 מחזורים, ביצוע sub/add לוקח מחזור אחד.

•ביצוע load לוקח 6 מחזורים לאחר שהכתובת מוכנה והblock_code=0-.

•עבור פקודת store, חישוב הכתובת לוקח מחזור אחד (ואז מוסרת חסימה של כתובת לא ידועה). חישוב המידע לוקח מחזור אחד. הביצוע מסתיים כאשר גם הכתובת וגם המידע שרוצים לכתוב מוכנים.

•פקודה יכולה לבצע commit אחרי המחזור שבו סימנו עבורה Data valid ב־ROB, בתנאי שהפקודה שלפניה ביצעה/מבצעת commit ולכל היותר 2 פקודות באותו מחזור.

•הניחו שגודל הROB 6 פקודות (בROB Cont מספיק מקום בשביל להכיל כמה חישובים שתצטרכו), בStore Buffer, Load Buffer כניסה אחת ושאר יחידות החישוב בלתי מוגבלות.

•חיזוי הקפיצות תמיד NT.

•בcommit של חיזוי שגוי עושים flush לכל רכיבי החומרה ובמחזור הבא יטענו פקודות מהכתובת הנכונה.

· שימו לב:

1. בפקודות הכוונה לפקודות שלמות ולא לחלקי Execute של פקודה אחת.
2. כל עוד הפקודה לא סיימה לעשות commit, היא עדיין במעבד.
3. לשם נוחות, אפשר לתת לפקודה שם לפי מספר השורה בה היא מופיעה בקוד לעיל.

<u>תשובה:</u>

מחזור	פקודות במעבד
1	1,2
2	1,2,3,5

1,2,3,5,6,7	3-4
2,3,5,6,7,8	5
5,6	6
5,6,7,8	7
5,6,7,8,10	8-12
6,7,8,10	13-18
6,7,8,10	19-20
6,7,8,10	21
7,8,10	22
7,8,10	23
8,10	24
8	25