BAR-ILAN UNIVERSITY (RA)

Faculty of Engineering

Ramat-Gan 52900, Israel

אוניברסיטת בר-אילן (ע"ר)

הפקולטה להנדסה

רמת-גן 52900

Tel: 03-5317722

engbi@mail.biu.ac.il

מבנה מחשבים ספרתיים

תש"ף סמסטר ב' מועד ב'

83-301

מרצה: פרופ' שמואל וימר

מתרגל: מר בנימין פרנקל

- **יש לקרוא היטב את ההוראות.**
- **חובה לענות על כל השאלות.**
- **ציון מקסימלי בבחינה: 100 נקודות.**
- **יש לנמק את כל תשובותיכם.**
- יש להקפיד על כתב יד קריא!
- יש לרשום תשובות בתוך הטבלאות המצורפות במקום שנדרש.
- **חומר עזר מותר בשימוש**: כל חומר עזר מודפס ומחשבון.
- **משך הבחינה:** שעתיים
- **יש לצרף את שאלוני הבחינה למחברת!**

בהצלחה!

<u>שאלה מספר 1 – Multi-threading (50 נקודות)</u>

נתונה מכונת Multi-thread ללא זיכרון מטמון העובדת ב- Switch-on-Event, אשר בה ה- CPU מחליף thread ברגע שקורה event מסוים הגורם לעיכוב משמעותי (לדוגמא: גישה לזיכרון). היות ואין במכונה זו זיכרון מטמון, כל פקודת גישה לזיכרון מוגדרת כ- event.

עבור כל ה- threads נתון כי כל פקודה שישית הינה פקודת גישה לזיכרון. כמו-כן, נתון כי הגישה לזיכרון אורכת 90 מחזורי שעון. בנוסף נתון כי $CPI_{ideal} = 1$, כאשר CPI_{ideal} איננו לוקח בחשבון את זמן הגישה לזיכרון.

א. בטאו את ביצועי המכונה במונחי *IPC* (*Instructions per cycle*) כתלות ב- *N* המציין את מספר ה- *threads* הקיימים במערכת. (8 נקודות)

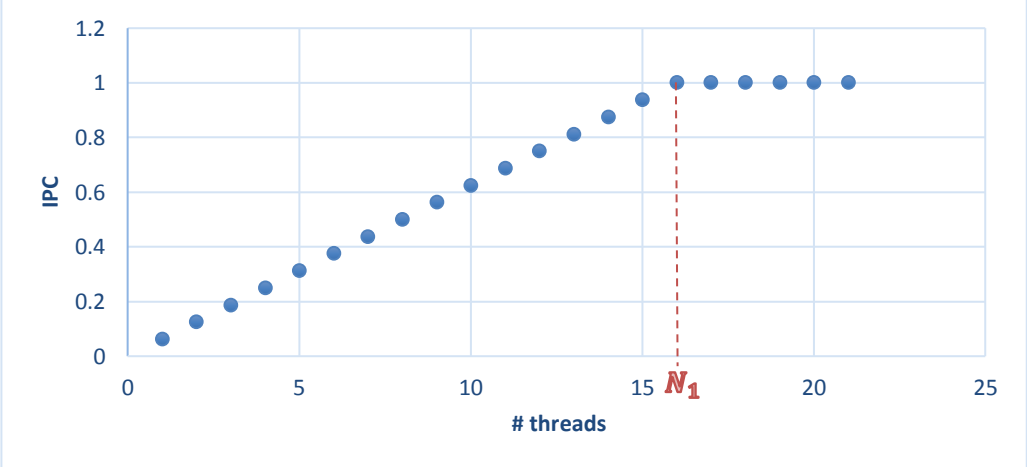
$$IPC = \begin{cases} \frac{1}{CPI_i} = 1 & , \quad N \geq N_1 \\ \frac{N \cdot 6}{6 \cdot CPI_i + 90} = N \cdot \frac{1}{16} & , \quad N < N_1 \end{cases}$$

כאשר N_1 מציין את מספר ה- *threads* המינימלי הנדרש על מנת להגיע לביצועים מקסימליים.

ב. מהם הביצועים המקסימליים אליהם ניתן להגיע (*IPC* מקסימלי)? (7 נקודות)

$$IPC_{max} = \frac{1}{CPI_i} = 1$$

ג. ציירו גרף המתאר את ה- *IPC* כפונקציה של *N* (מספר ה- *threads* במערכת). (6 נקודות)



ד. מהו מספר ה- *threads* המינימלי הנדרש על מנת להגיע לביצועים מקסימליים? סמנו את ערכו על גבי הגרף שציירתם בסעיף הקודם (ציינו אותו כ- N_1) וחשבו את ערכו על-פי הנתונים. (10 נקודות)

$N_1 = 16$

- כעת, הוצע שיפור במבנה המיקרו-ארכיטקטורה: הוספה של זיכרון מטמון (*cache*). הניחו שגישה לזיכרון המטמון אורכת אפס מחזורי שעון (זמן הגישה לזיכרון המטמון נכלל כבר ב- CPI_{ideal}).

ה. **נניח** שנתון כי זיכרון המטמון מחטיא בכל גישה עשירית לזיכרון. מהו מספר ה- threads המינימלי הדרוש כעת על מנת לשמור על ביצועים מקסימליים? הסבירו את תשובתכם! (10 נקודות)

כעת, כל *thread* מסוגל למלא את המעבד ב- 60 פקודות לפני שהוא מגיע ל- event. לכן, כדי למסך באופן מלא החטאה של *thread* מסוים, יש צורך לבצע 90 פקודות, ולכך נדרשים עוד 2 *threads*, ולכן סה"כ יש צורך ב- 3 *threads*.

ו. הסבירו באופן איכותי מה יהיו ההשלכות של הוספת זיכרון מטמון כמתואר בסעיף הקודם על המערכת מההיבטים הבאים: (9 נקודות סה"כ)

- מבחינת שטח

זה תלוי: מצד אחד, הוספת זיכרון מטמון תגדיל את השטח (זיכרון המטמון עצמו תופס שטח), ומצד שני, אם נוריד את מספר ה- *threads* שהחומרה תומכת בהם – נוכל להקטין את השטח הנצרך לטיפול ב- *threads* שונים (*RF* וכו').

- מבחינת הספק

זה תלוי: מצד אחד, זיכרון המטמון עצמו צורך הספק, ולכן ההספק יגדל. מצד שני, הוא חוסך לנו גישות לזיכרון, ולכן ההספק יקטן. באופן כללי, זה תלוי כמה גישות לזיכרון יש לנו בתוכנית וכמה זיכרון המטמון מחטיא.

- מבחינת סיבוכיות המערכת

זה תלוי: מצד אחד, הוספנו היררכיה לזיכרון, ולכן הסיבוכיות גדלה. מצד שני, הורדנו את מספר ה- *threads* שהמערכת צריכה לתמוך בהם, ולכן הסיבוכיות קטנה.

שאלה מספר 2 – OoO Execution (70 נקודות)

נתון מעבד בעל חומרה המבצעת OoO Execution בעזרת אלגוריתם Tomasulo עם ספקולציות (with speculations). נתונים הדברים הבאים:

- רק פקודה אחת מסוגלת לעשות issue בכל מחזור שעון
- ל- ROB ישנן 8 כניסות (slots), והוא משמש גם בתור חוצץ עבור פקודות load ו- store.
- כל היחידות הפונקציונליות (FUs) הינן fully pipelined.
- ישנן שתי FP reservation stations עבור פעולות כפל, ושלוש FP reservation stations עבור פעולות חיבור. כמו-כן, ישנן שלוש integer reservation stations אשר משרתות גם את פקודות ה- load וה- store.
- הניחו שלא מתרחשות חריגות (exceptions) במהלך ריצת התוכנית.
- זמן הביצוע (execution) של פעולות integer הוא מחזור שעון אחד. אם מדובר בפקודת load אזי הנח כי באותו מחזור שעון של חישוב הכתובת האפקטיבית מתבצעת גם הפניה והקריאה מהזיכרון. להוציא structural hazards, פקודות load עושות issue במחזור שעון אחד, מבצעות execution במחזור השעון השני, ומבצעות כתיבה (write) במחזור השעון השלישי, כך שפקודה שתלויה בתוצאה של פקודת ה- load יכולה להתחיל את שלב ה- execution שלה במחזור השעון הרביעי.
- זמן הביצוע (execution) של פעולות FP-multiply הינו 4 מחזורי שעון, וזמן הביצוע של פעולות FP-addition הינו 2 מחזורי שעון.
- כאשר ישנו conflict בכתיבה ל- CDB, ניתנת עדיפות (priority) לפקודה המוקדמת יותר.
- ביצוע של פקודה התלויה באופרנד יכול להתחיל במחזור השעון לאחר שהאופרנד הדרוש שודר על ה- CDB.
- הניחו כי כל ה- RS, FUs,ROB הינן ריקות ופנויות (not busy) בעת תחילת ביצוע התוכנית שלהלן.
- העמודה "value" מתעדכנת בעת שידור הערך על ה- CDB.

על המעבד מריצים את קטע הקוד הבא:

```
LD      F0,  0(R1)
LD      F2,  0(R2)
MULTD   F4,  F0,  F2
ADDD    F6,  F0,  F0
SUBI    R1,  R1,  8
SUBI    R2,  R2,  8
ADDI    R3,  R3,  1
```

הטבלאות הבאות מתארות את מצב החומרה לאחר מחזור השעון שבו פקודת ה- SUBI השנייה ביצעה את שלב ה- Issue.

א. מלאו את הטבלאות כך שיתארו את מצב החומרה בחלוף עוד מחזור שעון אחד. אם שורה מסוימת עוברת מ- Busy ל- Not Busy, עדכנו את ערך ה- Busy בעמודה המתאימה, אך אל תמחקו את הערכים בשאר העמודות באותה שורה (אלא אם כן פקודה אחרת נכנסת לאותו רגיסטר, ודורסת

את הערכים שהיו שם). Integer registers אינם מופיעים בטבלה האחרונה, ואינכם צריכים להתייחס לסטטוס שלהם. (20 נקודות)

Reservation Station							
Name	**Busy**	**Op**	**Vj**	**Vk**	**Qj**	**Qk**	**Dest**
Add1	Y	ADDD	F0	F0			#4
Add2							
Add3							
Mult1	Y	MULTD	F0	F2			#3
Mult2							
Int1	Y	SUBI	R2	8			#6
Int2	N	LD	R2	0			#2
Int3	Y	SUBI	R1	8			#5

Reservation Station							
Name	**Busy**	**Op**	**Vj**	**Vk**	**Qj**	**Qk**	**Dest**
Add1	N	ADDD	F0	F0			#4
Add2							
Add3							
Mult1	Y	MULTD	F0	F2			#3
Mult2							
Int1	Y	SUBI	R2	8			#6
Int2	Y	ADDI	R3	1			#7
Int3	Y	SUBI	R1	8			#5

Reorder Buffer					
Entry	**Busy**	**Instruction**	**State**	**Destination**	**Value**
1	N	LD F0, 0(R1)	commit	F0	Mem[0(R1)]
2	N	LD F2, 0(R2)	commit	F2	Mem[0(R2)]
3	Y	MULTD F4, F0, F2	execution	F4	
4	Y	ADDD F6, F0, F0	execution	F6	
5	Y	SUBI R1, R1, 8	execution	R1	
6	Y	SUBI R2, R2, 8	issue	R2	
7					
8					

Reorder Buffer					
Entry	**Busy**	**Instruction**	**State**	**Destination**	**Value**
1	N	LD F0, 0(R1)	commit	F0	Mem[0(R1)]
2	N	LD F2, 0(R2)	commit	F2	Mem[0(R2)]
3	Y	MULTD F4, F0, F2	execution	F4	
4	Y	ADDD F6, F0, F0	write	F6	F0 + F0
5	Y	SUBI R1, R1, 8	execution	R1	
6	Y	SUBI R2, R2, 8	execution	R2	
7	Y	ADDI R3, R3, 1	issue	R3	
8					

FP Register Status									
Field	**F0**	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**
Reorder #	1		2		3		4		
Busy	N		N		Y		Y		

FP Register Status									
Field	**F0**	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**
Reorder #	1		2		3		4		
Busy	N		N		Y		Y		

ב. נגדיר את מחזור השעון שבו פקודת ה- LD הראשונה ביצעה issue. באיזה מחזור שעון תבצע הפקודה האחרונה ברצף הפקודות את שלב ה- commit? (7 נקודות)

במחזור שעון מספר 14.

ג. מהנדס צעיר רצה לבחון את ביצועי המעבד המוזכר בתחילת השאלה לביצועי מעבד אשר מבצע את אלגוריתם Tomasulo, אך איננו תומך בספקולציות (dynamic scheduling, no speculations). מה תהיה ההשפעה על הביצועים אם נריץ את קטע הקוד הנ"ל על מעבד כזה (ללא ROB)? חשבו את ה- speedup. (7 נקודות)

הפקודה האחרונה תבצע WB במחזור שעון מספר 11, ולכן: $\text{speedup} = \frac{14}{11} \cong 1.27$

- כעת, על אותו מעבד המתואר בתחילת השאלה מריצים את קטע הקוד הבא:

```
MULTD   F0,   F2,   F4
ADDD    F6,   F6,   F0
ADDD    F2,   F2,   F8
LD      F4,   0(R2)
ADDI    R1,   R1,   8
MULTD   F8,   F10,  F12
ADDD    F4,   F4,   F10
ADDI    R2,   R2    1
```

הטבלאות הבאות מתארות את מצב החומרה במהלך מחזור השעון שבו פקודת ה- MULT השנייה מבצעת את שלב ה- Issue.

ד. מלאו את הטבלאות כך שיתארו את מצב החומרה בחלוף עוד שני מחזורי שעון. אם שורה מסוימת עוברת מ- Busy ל- Not Busy, עדכנו את ערך ה- Busy בעמודה המתאימה, אך אל תמחקו את הערכים בשאר העמודות באותה שורה (אלא אם כן פקודה אחרת נכנסת לאותו רגיסטר, ודורסת את הערכים שהיו שם). Integer registers אינם מופיעים בטבלה האחרונה, ואינכם צריכים להתייחס לסטטוס שלהם. (20 נקודות)

Reservation Station							
Name	**Busy**	**Op**	**Vj**	**Vk**	**Qj**	**Qk**	**Dest**
Add1	Y	ADDD	F6			#1	#2
Add2	Y	ADDD	F2	F8			#3
Add3							
Mult1	N	MULTD	F2	F4			#1
Mult2	Y	MULTD	F10	F12			#6
Int1	Y	LD	R2	0			#4
Int2	Y	ADDI	R1	8			#5
Int3							

Reservation Station							
Name	**Busy**	**Op**	**Vj**	**Vk**	**Qj**	**Qk**	**Dest**
Add1	Y	ADDD	F6	F0		~~#1~~	#2
Add2	N	ADDD	F2	F8			#3
Add3	Y	ADDD	F4	F10			#7
Mult1	N	MULTD	F2	F4			#1
Mult2	Y	MULTD	F10	F12			#6
Int1	N	LD	R2	0			#4
Int2	Y	ADDI	R1	8			#5
Int3	Y	ADDI	R2	1			#8

Reorder Buffer					
Entry	**Busy**	**Instruction**	**State**	**Destination**	**Value**
1	Y	MULTD F0, F2, F4	write	F0	F2×F4
2	Y	ADDD F6, F6, F0	issue	F6	
3	Y	ADDD F2, F2, F8	execution	F2	
4	Y	LD F4, 0(R2)	execution	F4	
5	Y	ADDI R1, R1, 8	execution	R1	
6	Y	MULTD F8, F10, F12	issue	F8	
7					
8					

Reorder Buffer					
Entry	**Busy**	**Instruction**	**State**	**Destination**	**Value**
1	N	MULTD F0, F2, F4	commit	F0	F2×F4
2	Y	ADDD F6, F6, F0	execution	F6	
3	Y	ADDD F2, F2, F8	write	F2	F2+F8
4	Y	LD F4, 0(R2)	write	F4	Mem[0(R2)]
5	Y	ADDI R1, R1, 8	execution	R1	
6	Y	MULTD F8, F10, F12	execution	F8	
7	Y	ADDD F4, F4, F10	issue	F4	
8	Y	ADDI R2, R2, 1	issue	R2	

FP Register Status									
Field	**F0**	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**
Reorder #	1		3		4		2		6
Busy	Y		Y		Y		Y		Y

FP Register Status									
Field	**F0**	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**
Reorder #	1		3		7		2		6
Busy	N		Y		Y		Y		Y

ה. מנו את כל סוגי ה- data hazards הקיימים במעבד Out-of-Order, והסבירו כיצד אלגוריתם Tomasulo מטפל בכל אחד מהם (די במשפט או שניים עבור כל סוג של data hazard). (6 נקודות)

RAW hazards – אלו תלויות מידע אמתיות. אלגוריתם Tomasulo מטפל בהם ע"י מעקב אחר התלויות במהלך שלב ה- issue, כך שפקודה שיש לה תלות של מידע בפקודה אחרת מקבלת את ה- tag של הפקודה שמייצרת את המידע. כאשר המידע המבוקש משודר על ה- CDB, כל הפקודות שזקוקות לאותו מידע קוראות אותו מה- CDB ע"י זיהוי של ה- tag.

WAR hazards – תלויות אלו מחוסלות באופן מוחלט, משום שהערכים מה- Register File מועתקים ל- RS בשלב ה- issue. כך, איך אפשרות שפקודה מאוחרת יותר תדרוס אופרנדים הנדרשים לקריאה ע"י פקודה מוקדמת יותר.

WAW hazards – התלויות הללו נמנעות ע"י המכניזם של register renaming, משום שה- Register File מקבל רצף של tags לפי סדר התוכנית שנכתבים אליו. אם יש שתי פקודות שכותבות לאותו Register שנמצאות במקביל ב- Pipeline, הרי שרק ה- tag של הפקודה **האחרונה** נשמר ב- Register File, וכך, אין סכנה שפקודה חדשה תקרא ערך ישן ולא עדכני.

ו. באלגוריתם Tomasulo ישנו "באג" מעניין: הניחו סיטואציה שבה פקודה אחת משתמשת בערך הנוצר ע"י פקודה אחרת (הקודמת לה). הניחו כי הפקודה המאוחרת מבצעת את שלב ה- Issue באותו מחזור שעון שבו הפקודה המוקדמת מבצעת את שלב ה- writeback שלה. לדוגמא:

```
ADD  $r1, $r2, $r3      ← The result is broadcast
...
ADD  $r4, $r1, $r1      ← This one is being issued
```

הסבירו את הבעיה בסיטואציה הזאת. האם תוכלו להציע פתרון יעיל ופשוט על מנת לפתור את הבעיה? (10 נקודות)

הבעיה היא שבתחילת מחזור שעון הלוגיקה של ה- issue תבדוק ב- Register File ותחליט שה-value עדיין לא מוכן, ולכן תשלח את הפקודה ל- RS עם tag מתאים. בינתיים, בסוף מחזור השעון, הערך החדש ייכתב אל ה- Register File. כעת, ישנו tag ב- RS וישנו value ב- RF, והפקודה החדשה שמחכה ל- value לעולם לא תתחיל את שלב ה- execution.

הפתרון הפשוט לבעיה הזאת הוא לבצע את ה- writeback בחצי מחזור שעון הראשון, ולהסתכל ב- RF בחצי מחזור השעון השני (אותו פתרון שבו השתמשנו ב- 5-stage pipeline).