תרגיל בית 2

שאלה 1

א. כאשר המערכת תעבוד הכי לאט, כלומר כאשר $v_{cc}=3v$, $T_A=60°c$, יש לבדוק שתנאי *Setup* מתקיימים, ולפי כך להבטיח תדר עבודה מכסימלי.

ב. כאשר המערת תעבוד הכי מהר, כלומר כאשר $v_{cc}=15v$, $T_A=-30°c$, ולפי כך להבטיח שתנאי *Hold* מתקיימים.

שאלה 2

סעיף א, $T_L << T_H$, כאילו קיים $T_{Skew}=T_L$ ושני הדלגלגים פעילים בעליית שעון

המציאות הנוצרת: $FF2$ מקדים את $FF1$

הדרישות הנכונות הן:

כדי שכניסת $FF1$ תהיה מוכנה בזמן כשהוא דוגם, נדרוש: $T_{pd}(FF2)+T_{pd}(CL2)+T_{Setup}(FF1)\le T_{cyc}+T_L$

כדי ש $FF1$ יספיק לדגום את מה ש $FF2$ הוציא לפני שדגם, נדרוש: $T_{Hold}(FF1)+T_L\le T_{cd}(FF2)+T_{cd}(CL2)$

כדי שכניסת $FF2$ תהיה מוכנה בזמן כשהוא דוגם, נדרוש: $T_{pd}(FF1)+T_{pd}(CL1)+T_{Setup}(FF2)\le T_H$

כדי ש $FF2$ יספיק לדגום את מה ש $FF1$ הוציא לפני שדגם, נדרוש: $T_{Hold}(FF2)\le T_L+T_{cd}(FF1)+T_{cd}(CL1)$

כלומר התנאים בשאלה נכונים, בתוספת קלה.

סעיף ב, $T_H << T_L$, כאילו קיים $T_{Skew}=T_H$ ושני הדלגלגים פעילים בעליית שעון

המציאות הנוצרת: $FF1$ מקדים את $FF2$

הדרישות הנכונות הן:

כדי שכניסת $FF1$ תהיה מוכנה בזמן כשהוא דוגם, נדרוש: $T_{pd}(FF2)+T_{pd}(CL2)+T_{Setup}(FF1)\le T_L$

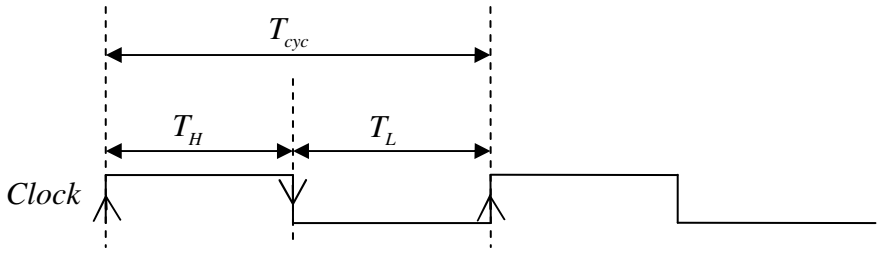
כדי ש $FF1$ יספיק לדגום את מה ש $FF2$ הוציא לפני שדגם, נדרוש: $T_{Hold}(FF1)\le T_H+T_{cd}(FF2)+T_{cd}(CL2)$

כדי שכניסת $FF2$ תהיה מוכנה בזמן כשהוא דוגם, נדרוש: $T_{pd}(FF1)+T_{pd}(CL1)+T_{Setup}(FF2)\le T_{cyc}+T_H$

כדי ש $FF2$ יספיק לדגום את מה ש $FF1$ הוציא לפני שדגם, נדרוש: $T_{Hold}(FF2)+T_H\le T_{cd}(FF1)+T_{cd}(CL1)$

כלומר התנאים בשאלה נכונים, בתוספת קלה.

הקדמה לסעיפים ג' וד', $T_L \cong T_H$, כלומר $FF1$ פעיל בעלית שעון, $FF2$ פעיל בירידת שעון:



השאלה בסעיפים אלו לא מוגדרת היטב ולכן, ישנן שלוש סוגי מציאות שיכולות להיות:

1. אין מקדים או מאחר, כל דלגלג צריך לעבוד עם הערך החדש שמחושב ע"י הדלגלג שלפניו.
   הדרישות הנכונות לפעולה תקינה של המערכת:
   כדי שכניסת $FF1$ תהיה מוכנה בזמן כשהוא דוגם, נדרוש: $T_{pd}(FF2)+T_{pd}(CL2)+T_{Setup}(FF1)\le T_L$
   כדי שכניסת $FF2$ תהיה מוכנה בזמן כשהוא דוגם, נדרוש: $T_{pd}(FF1)+T_{pd}(CL1)+T_{Setup}(FF2)\le T_H$
   בהנחה ש $T_{Hold}\le T_{pd}$ לכל דלגלג, אין צורך בהגבלות נוספות על זמני T_{Hold}.

2. $FF1$ מקדים את $FF2$:
   נדרוש את אותן 4 הדרישות שבסעיף ב'

3. $FF2$ מקדים את $FF1$
   נדרוש את אותן 4 הדרישות שבסעיף א'

סעיף ג:

כל אחת משלוש האופציות (שלוש סוגי מציאות) שמתוארת לעיל יוצרת סתירה עם אחת הדרישות שבסעיף זה, ולכן סעיף זה חסר משמעות.

דוגמא לסתירה:

$T_{hold}(FF1)+T_L \le T_{cd}(FF2)+T_{cd}(CL2)$ יחד עם התנאי $T_{pd}(FF2)+T_{pd}(CL2)+T_{Setup}(FF1)\le T_L$ יביאו לתנאי:

$$T_{pd}(FF2)+T_{pd}(CL2)+T_{Setup}(FF1)\le T_L \le T_{cd}(FF2)+T_{cd}(CL2)-T_{hold}(FF1)$$

שלא יכול להתקיים לעולם בהנחה ש $T_{pd}, T_{Setup} < T_{cd}, T_{Hold}$ לכל דלגלג.

סעיף ד:

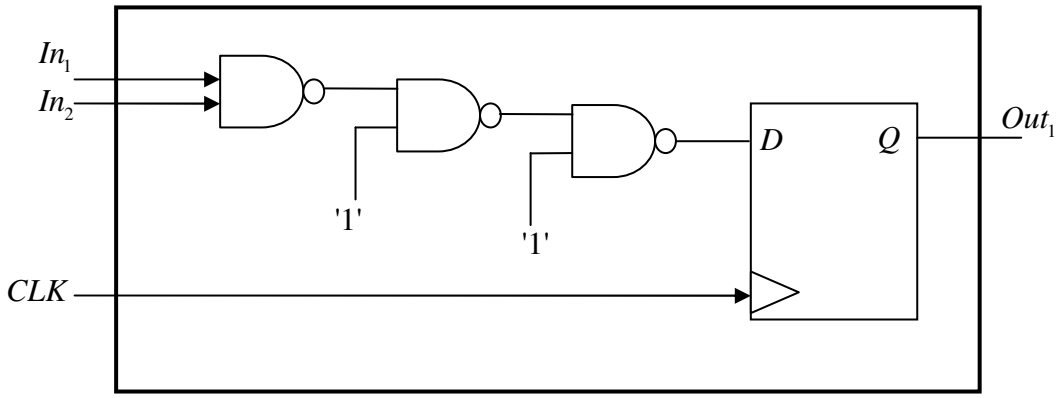
אין בעיה עם הדרישות שבסעיף, רק שיש להוסיף את הדרישות המתאימות, לפי הפירוט שלעיל.
כלומר התנאים בשאלה נכונים, בתוספת קלה.

שאלה 3

א. מאפייני הרכיב החדש:

t_{pd}	40
t_{cd}	5
t_{setup}	$20+22=42$
t_{hold}	$5-2=3$

ב. ניתן, ע"י הוספת שערי $NAND$, כשנעזר בזהות $NAND(x,'1') = \overline{x}$



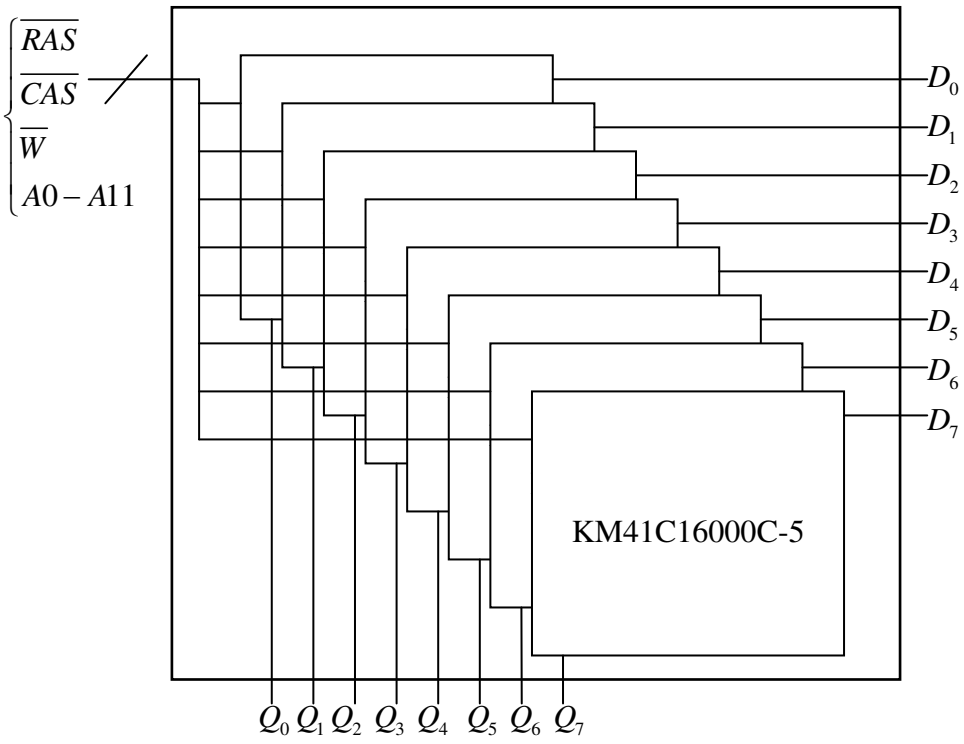
במצב הזה יובטח לנו $T_{hold} = 5-2-2-2 = -1$, תודות ל T_{cd} של ידידנו ה $NAND$.

תרגיל בית 3

שאלה 1

סעיף א

נידרש ל 8 רכיבי KM41C16000C-5 , כ"א מהם מחזיק ביט יחיד, כלומר שמינית מהמילה המבוקשת בכל פעם.



סעיף ב

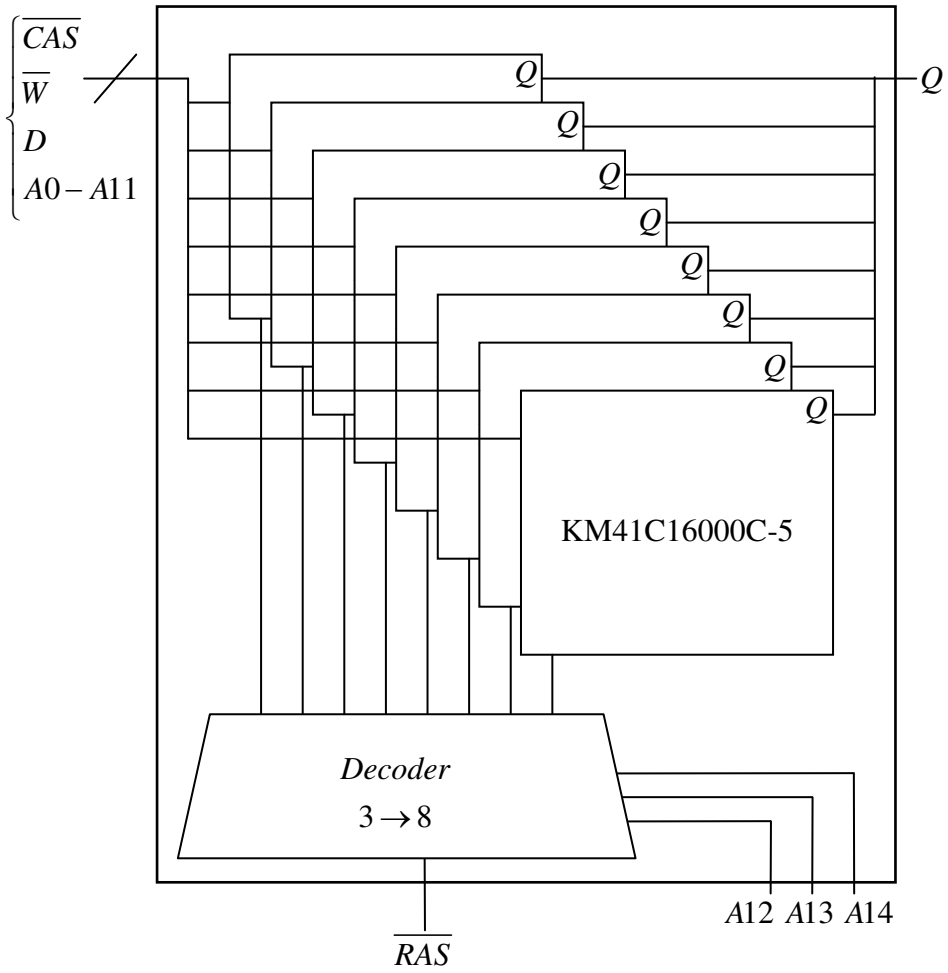
בהנחה שמימוש זה לא פוגם ביכולות ה Fanout של הכניסות לרכיב החדש שלנו, תזמוני אותות הבקרה לא משתנים. בכל מקרה, תזמוני כניסות ויציאות הנתונים לא ישתנו.

סעיף ג

נדרש ל 8 רכיבי DRAM מהסוג המדובר. שלושת הביטים העליונים של הכתובת משמשים לבחירת רכיב ה DRAM הפעיל, אליו יכנס אות RAS ולכן חייבים להיות פעילים לאורך מחזור הגישה.

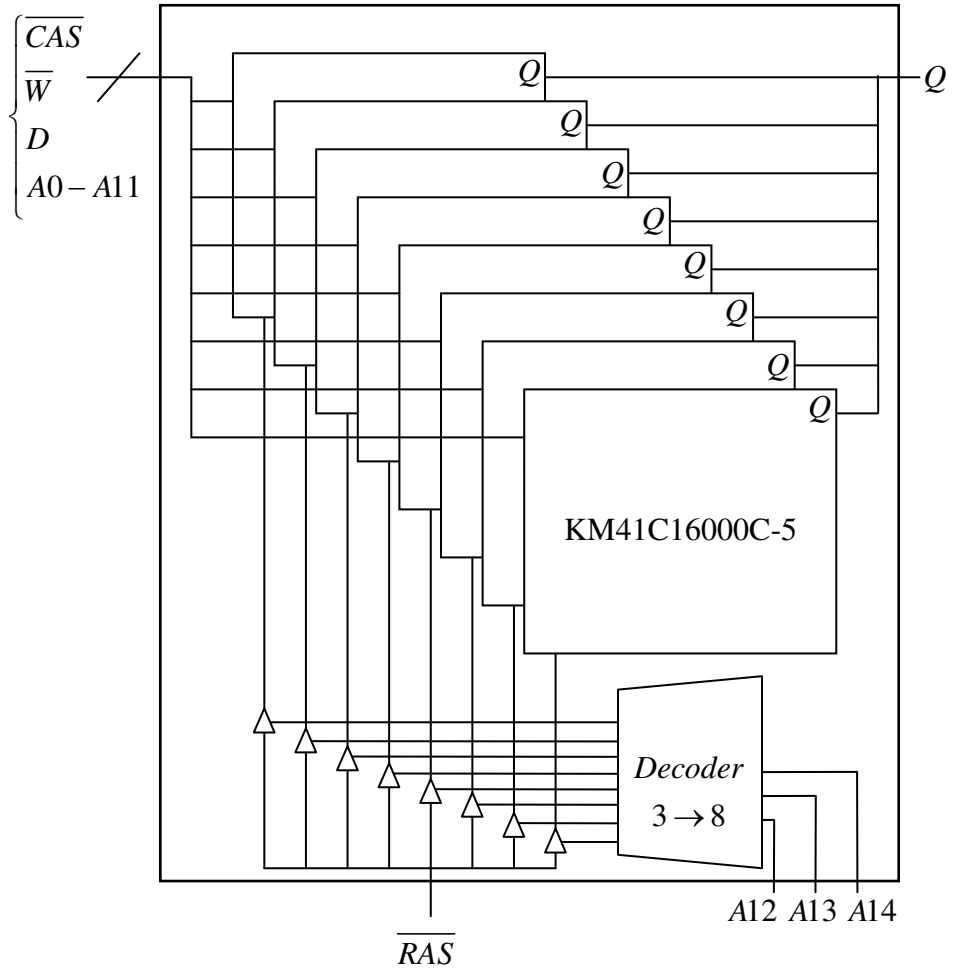
סעיף ד

הזמנים של הרכיב החדש שיצרנו יושפעו מזמני שיהוי של המפענח. האות RAS יגיע אל רכיב ה DRAM הפנימי הפעיל באיחור $Tpd_{in\to out}\left(Decoder\right)$, שנסמן בקיצור Tpd, כאשר שלושת הביטים העליונים של הכתובת יציבים.



עקב כך, נקבל זמנים השהייה שונים עבור הרכיב שלנו:

$$t_{RRH}^{*}=t_{RRH}+Tpd$$
$$t_{RAH}^{*}=t_{RAH}+Tpd$$
$$t_{CRP}^{*}=t_{CRP}-Tpd$$
$$t_{RAL}^{*}=t_{RAL}-Tpd$$
$$t_{RAC}^{*}=t_{RAC}+Tpd$$
$$t_{CAC}^{*}=t_{CAC}+Tpd$$



<u>סעיף ה</u>

נעשה דבר פשוט – שלושת הביטים העליונים של כתובת השורה יכנסו למפענח, שיבחר מי מבין שמונת החוצצים פעיל. ברגע שחוצץ אחד מתוך השמונה פעיל, אות ה RAS הנכנס לרכיב שלנו יעבור אל ה DRAM הפעיל.

תזמוני אותות הבקרה RAS , CAS ו W לא משתנים, אלא רק תזמון אות כתובת השורה:

$$t_{RAH}{}^{*} = t_{RAH} + Tpd$$

<u>סעיף ו</u>

הרענון בקריאה יתבצע ללא קשר להשהיות ה RAS , אם קיימות, ולכן אין יתרון לאחד המימושים בהקשר זה.

<u>סעיף ז</u>

הנחנו שה $t_{PD} = 0$ עבור שמונת החוצצים.

שאלה 2

סעיף א

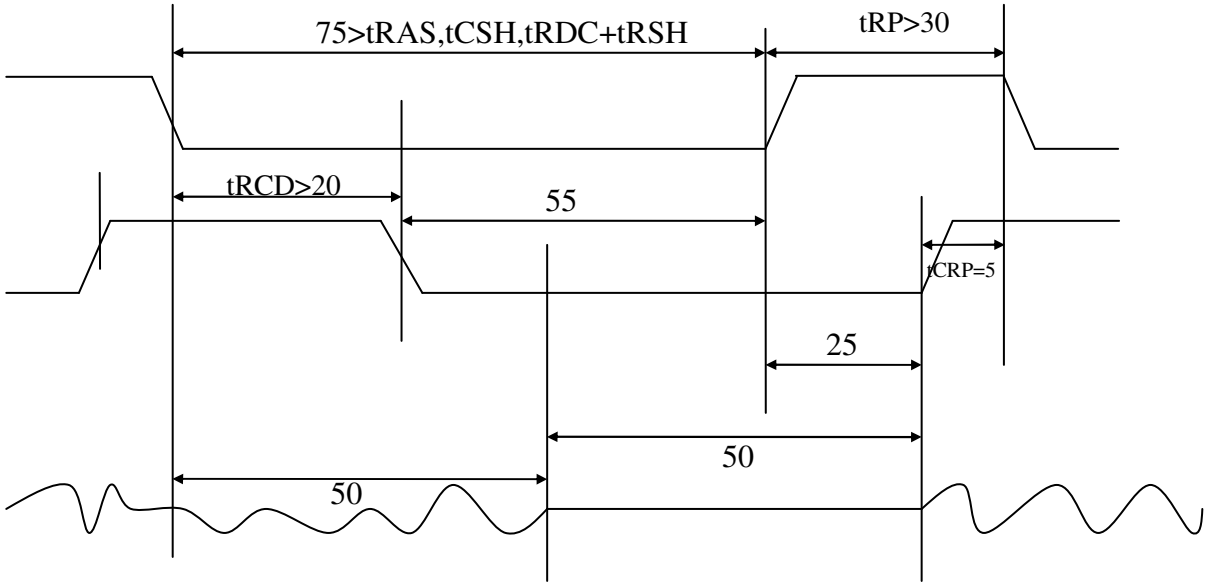
לא ניתן לתזמן תזמון מינמלי את אות ה CAS, כי אז $t_{RCD}+t_{CAS}=20+13=33$, ולעומת זאת מובטחת יציאה יציבה רק $t_{RAC}=50$ לאחר ירידת RAS. במצב זה, כלל לא ניתן להבטיח יציאה יציבה.

סעיף ב

נבדוק מסלולים שונים:

1. $T_{\min}\geq t_{RC}=90ns$
2. $T_{\min}\geq t_{RAS}+t_{RP}=50+30=80ns$
3. $T_{\min}\geq \max\{t_{CRP},t_{ASR}\}+\max\{t_{RAC\max},t_{RAD}+t_{AA\max},t_{RCD\min}+t_{CAC\max}\}+50-t_{OFF\min}=$
$=\max\{5,0\}+\max\{50,15+25,20+13\}+50-0=5+50+50=105ns$

ולסיכום, $T_{\min}\geq 105ns$



סעיף ג

בהנחה שהמחזוריות של הנתון ביציאה לא חייבת להיות זהה למחזוריות הקריאה - כלומר במצב של זמן מחזור של $95ns$, הנתון יהיה יציב $45ns$ במחזור הנוכחי ועוד $5ns$ במחזור הבא. ואז, בנוסף לדרישות (1) ו (2) של הסעיף הקודם, נקבל את הדרישה הזו:

$$T_{\min}=\max\{t_{CRP},t_{ASR}\}+\max\{t_{RAC\max},t_{RAD}+t_{AA\max},t_{RCD\min}+t_{CAC\max}\}+50-t_{OFF\min}=$$
$$=\max\{5,0\}+\max\{50,15+25,20+13\}+50-10=$$
$$=5+50+50-10=95ns$$

ולכן לסיכום, $T_{\min}\geq 95ns$

אך, אם מחזוריות הנתון צריכה להתאים למחזור הקריאה, לא נוכל לרדת מזמן מחזור של $T_{\min}\geq 100ns$.

שאלה 3

סעיף א

בהנחה שנשאיר את CAS גבוה לאורך כל הרענון:

$$t = \max\{t_{CRP}, t_{ASR}\} + [rows] \cdot \max\{(t_{RAS\min} + t_{RP\min}), t_{RC}\} =$$
$$= \max\{5ns, 0\} + 2^{12} \cdot \max\{(50ns + 30ns), 90ns\} =$$
$$= 368.645\mu s \cong 0.37ms$$

סעיף ב

בהנחה שנשאיר את CAS נמוך ואת W גבוה לאורך כל הרענון:

$$t = \max\{t_{RP}, t_{CSR} + \max\{t_{RPC}, t_{CP}\}\} + [rows] \cdot \max\{(t_{RAS\min} + t_{RP\min}), t_{RC}\} =$$
$$= \max\{30, 5 + \max\{5,10\}\} + 2^{12} \cdot \max\{(50+30), 90\} =$$
$$= 30ns + 2^{12} \cdot 90ns = 368.67\mu s \cong 0.37ms$$

סעיף ג

הקריטריון הוא הזמן הפנוי הנשאר לך בעבודה עם ה DRAM, מתוך מחזור של $64ms$ שבו כל השורות צריכות להתרענן.
אם יש לך מספיק זמן פנוי, ניתן להריץ CAS BEFORE RAS למשך $0.4ms$, מה שיבטיח רענון של כל השורות, בלי לדאוג לכתובות השורה המתרעננות.
אם אין מרווח זמן פנוי להרצת רענון שכזה, נצטרך לעקוב אחרי השורות הפעילות שלנו, ולרענן את השורות המתאימות בעצמנו.

סעיף ד

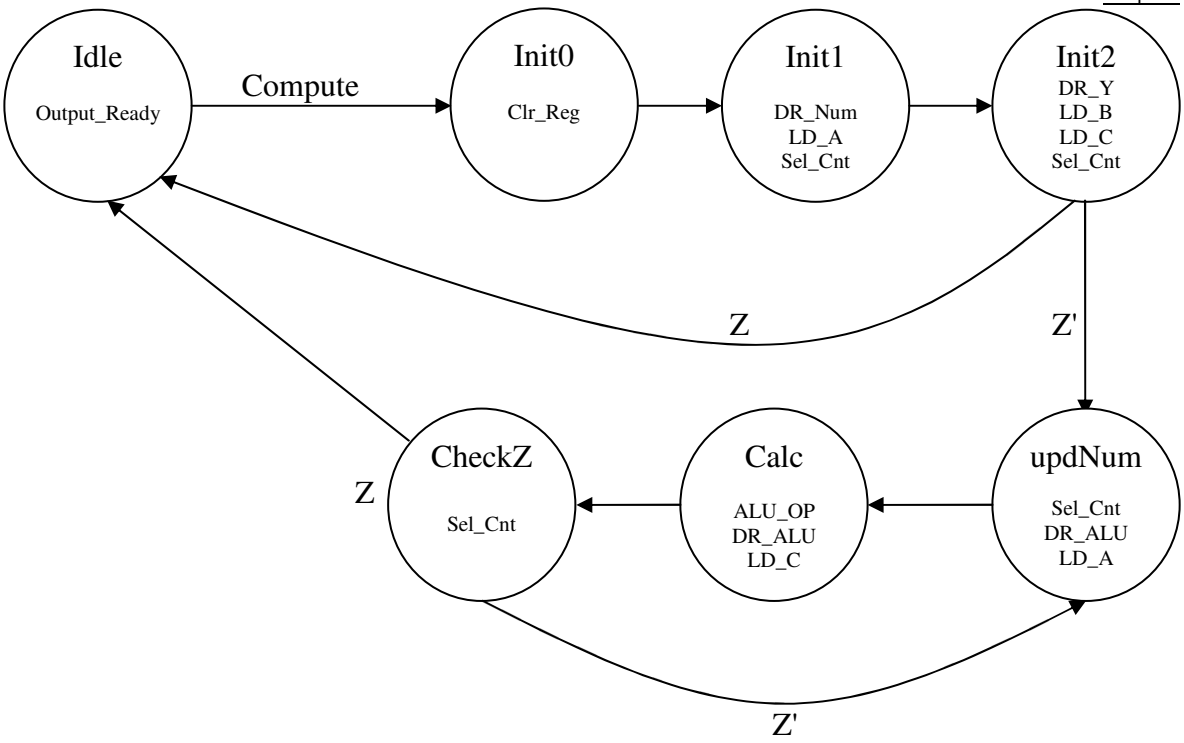
נשים לב שבזמן לולאה אחת של התוכנית שלנו, ניתן לרענן את כל ה DRAM כ-10 פעמים. רוב הלולאה, תוכניתנו "נחה" ולכן זמן של $0.4ms$ זניח ביחס לזמן הלולאה כולה, $5ms$. בגלל שהתוכנית מרענת בצורה קבועה 100 שורות בלבד לכל לולאה, הזמן שידרש לרענן בשיטת RAS ONLY אמנם יקטן ב $9000ns$, אך ידרוש מאיתנו משאבי חומרה נוספים, מה שלא ישפר בהכרח את ה TRADEOFF בין חסכון בזמן רענון ה DRAM וסיבוך המימוש של ביצוע הרענון.
בנוסף,
בהנחה שהתוכנית שלנו היא היחידה, ושם תוכנית אחרת לא משתמשת או מרעננת את ה DRAM, אז מהשיקול של הסעיף הקודם, הכי פשוט יהיה להריץ רענון CAS BEFORE RAS, שיגזול לנו כ $0.4ms$, כל 10 לולאות שלנו.

אם היינו ניגשים באופן תדיר ליותר מ-100 שורות בלולאה אחת, אפשרי שהזמן הנחסך ע"י RAS ONLY, כנגד CBR, היה משמעותי בשבילנו.

תרגיל בית 4

שאלה 1

סעיף א



סה"כ 7 מצבים. טבלת המצבים:

הסבר	שם	דצימלי	קידוד
המערכת נחה	Idle	0	000
איפוס האוגרים	Init0	1	001
הטענת Num	Init1	2	010
הטענת Y	Init2	3	011
הטענת Num-1 ל A	updNum	4	100
הטענת B+C ל C	Calc	5	101
בדיקה האם Num-1=0	CheckZ	6	110

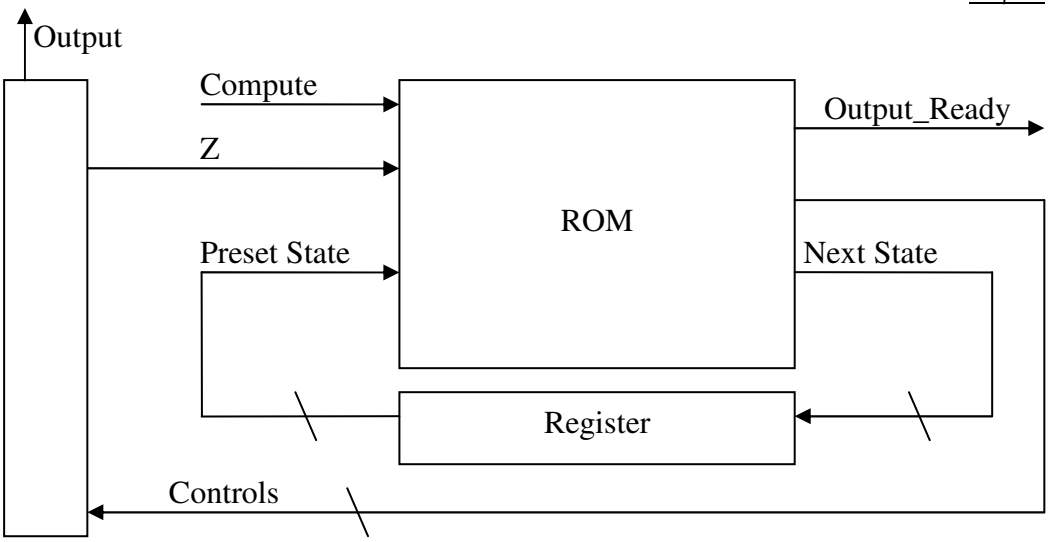
יציאות הבקר:

OutRdy	Clr_Reg	Sel_Cnt	ALU_OP	DR_Y	DR_Num	DR_ALU	LD_C	LD_B	LD_A	מצב
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	1	0	0	0	1	2
0	0	1	0	1	0	0	1	1	0	3
0	0	1	0	0	0	1	0	0	1	4
0	0	0	1	0	0	1	1	0	0	5
0	0	1	0	0	0	0	0	0	0	6

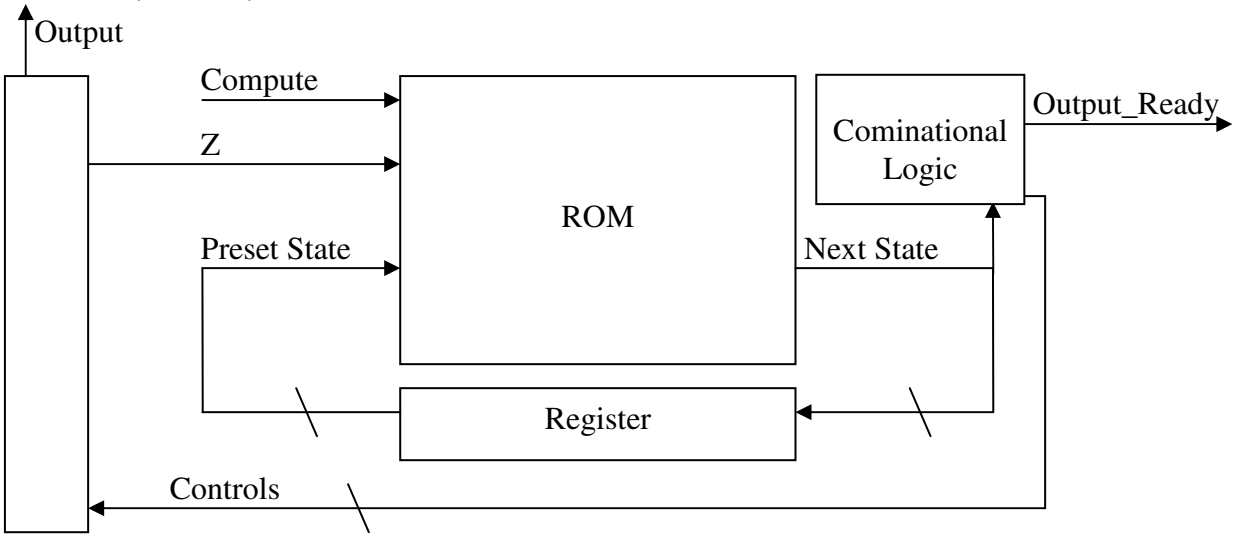
חישוב המצב הבא:

PS	Compute=1	
	Z=0	Z=1
000	001	001
001	010	010
010	011	011
011	100	000
100	101	101
101	110	110
110	100	000

סעיף ב



או, ניתן לחשב את קווי הבקרה בעזרת שלושת הביטים של המצב הבא, בעזרת לוגיקה נוספת (ראה נספח) ולקבל:



סעיף ג

ל ROM יש 5 ביטים עבור כתובת: 3 עבור המצב הנוכחי, Compute ו Z.

כל כתובת של ה ROM צריכה להחיל 3 ביטים עבור המצב הבא, ביט Output_Ready ו 9 קווי בקרה נוספים.

סה"כ 2^5 כתובת ברוחב 13 כלומר $2^5 \times 13 = 416$ סיביות של ROM.

כדי לחסוך בזכרון ROM, ניתן לשמור רק את המצב הבא, ולחשב לפיו את קווי הבקרה, בעזרת לוגיקה נוספת (ראה נספח).

כך נקבל ROM בגודל של: $2^5 \times 3 = 96$.

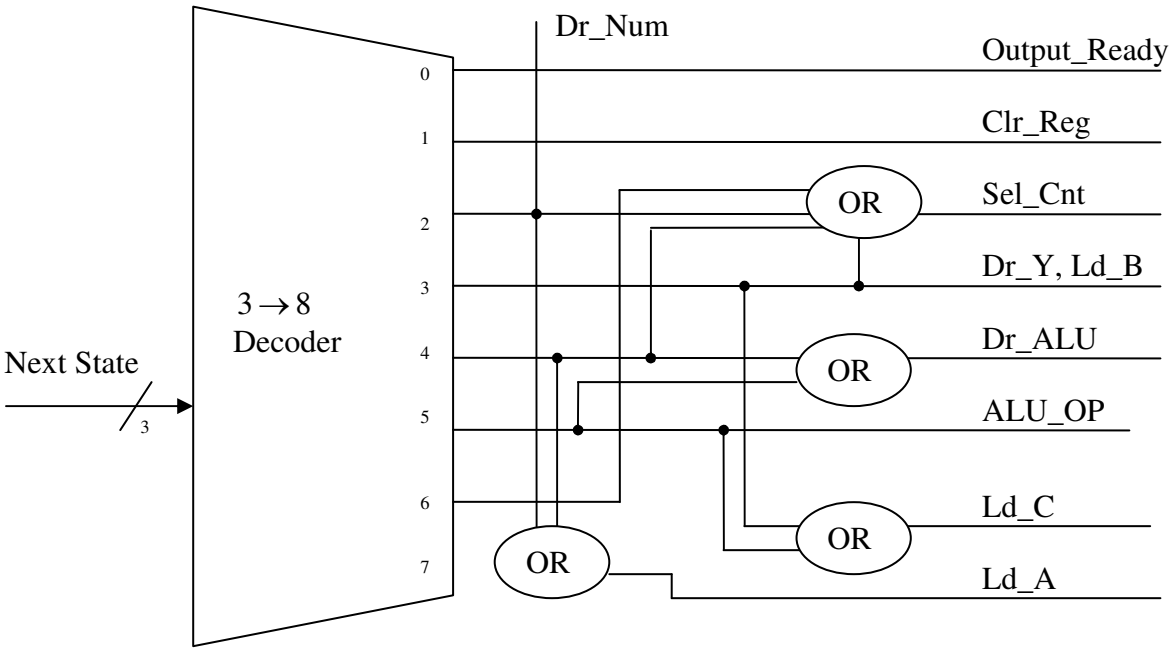
סעיף ד

Address			Data										
PS	Compute	Z	OutRdy	ClrReg	SelCnt	ALUOP	DR_Y	DR_Num	DRALU	LD_C	LD_B	LD_A	NS
000	0	ϕ	1	0	0	0	0	0	0	0	0	0	000
000	1	ϕ	1	0	0	0	0	0	0	0	0	0	001
001	ϕ	ϕ	0	1	0	0	0	0	0	0	0	0	010
010	ϕ	ϕ	0	0	1	0	0	1	0	0	0	1	011
011	ϕ	0	0	0	1	0	1	0	0	1	1	0	100
011	ϕ	1	0	0	1	0	1	0	0	1	1	0	000
100	ϕ	ϕ	0	0	1	0	0	0	1	0	0	1	101
101	ϕ	ϕ	0	0	0	1	0	0	1	1	0	0	110
110	ϕ	0	0	0	1	0	0	0	0	0	0	0	100
110	ϕ	1	0	0	1	0	0	0	0	0	0	0	000
111	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ

אם קווי הבקרה מחושבים משלושת הביטים של המצב הבא, טבלת ה ROM תהיה מצומצמת יותר:

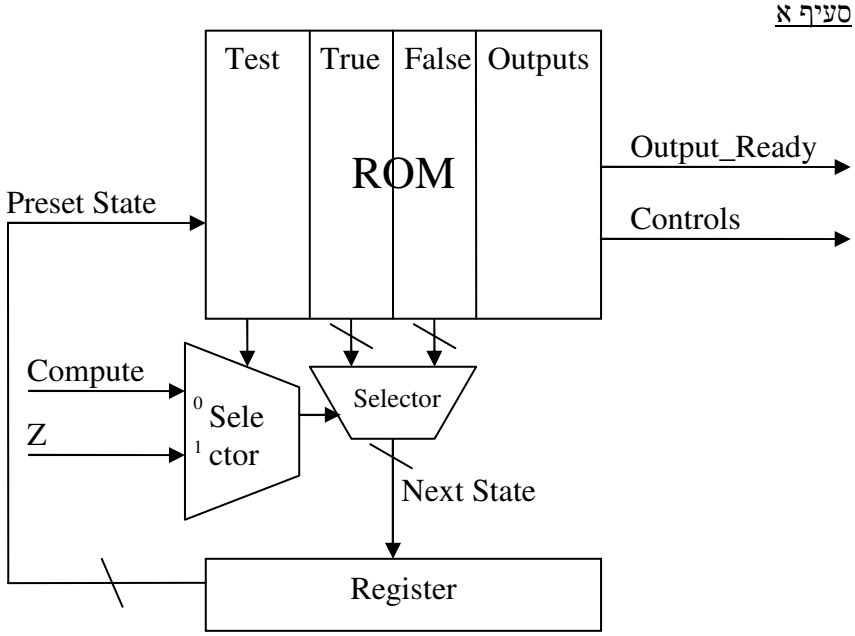
Addres			Data
PS	Compute	Z	NS
000	0	ϕ	000
000	1	ϕ	001
001	ϕ	ϕ	010
010	ϕ	ϕ	011
011	ϕ	0	100
011	ϕ	1	000
100	ϕ	ϕ	101
101	ϕ	ϕ	110
110	ϕ	0	100
110	ϕ	1	000
111	ϕ	ϕ	ϕ

נספח – הלוגיקה הנוספת הנדרשת לחישוב קווי הבקרה, כאשר משתמשים בשלושת הביטים של המצב הבא:

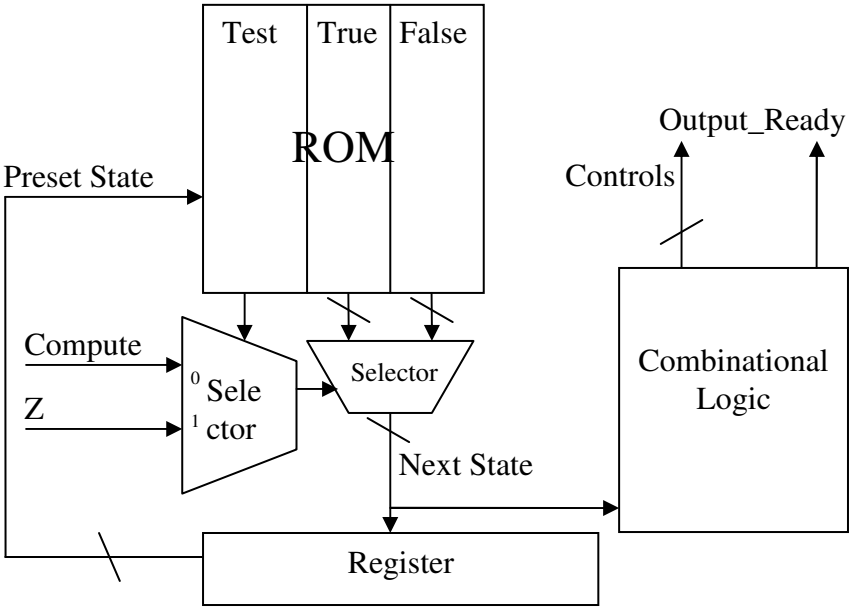


שאלה 2

סעיף א



כמו בשאלה הקודמת, ניתן להוסיף לוגיקה (ראה נספח בשאלה 1) לחישוב קווי הבקרה בעזרת שלושת הביטים של המצב הבא, ואז לקבל:



סעיף ב

כעת אנו זקוקים רק ל 2^3 כתובות.

רוחב כל כתובת: ביט בודד ל Test, כי יש לנו שני משתנים; 3 ביטים לעמודת True ולעמודת False כ"א; 10 ביטים של בקרה בעמודת Outputs.

סיכום: גודל ה ROM: $2^3 \times (1+3+3+10) = 2^3 \times 17 = 136$

אם נשתמש בלוגיקה הנוספת לחישוב קווי הבקרה, נזדקק רק לעמודות ה test, linkT, linkF, ולכן נקבל ROM קטן יותר:

$2^3 \times 7 = 56$

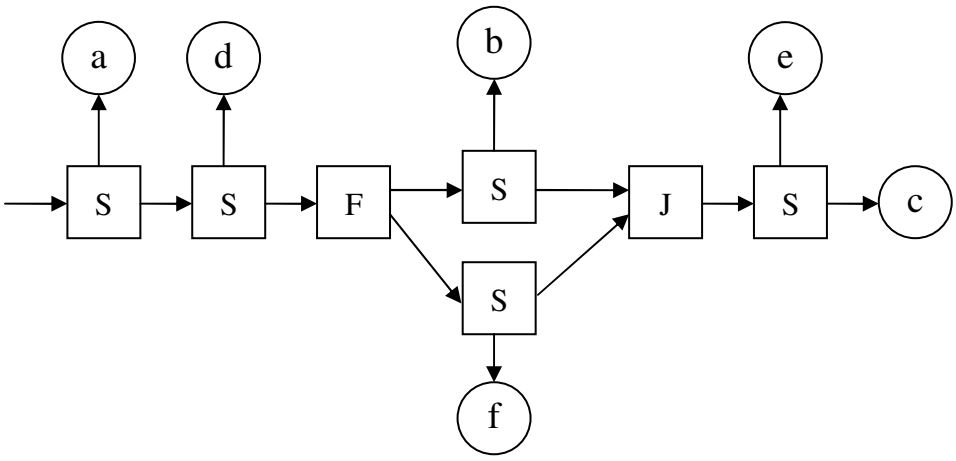
סעיף ג

Address	test	linkTrue	linkFalse	Output
000	0	001	000	1000000000
001	ϕ	010	010	0000000000
010	ϕ	011	011	0100000000
011	1	000	100	0010010001
100	ϕ	101	101	0010001001
101	ϕ	110	110	0001001100
110	1	000	100	0010000000
111	ϕ	ϕ	ϕ	ϕ

אם לא נחזיק ב ROM את קווי הבקרה, תוכנו יהיה:

Address	test	linkTrue	linkFalse
000	0	001	000
001	ϕ	010	010
010	ϕ	011	011
011	1	000	100
100	ϕ	101	101
101	ϕ	110	110
110	1	000	100
111	ϕ	ϕ	ϕ

שאלה 3



זמן החישוב של הבקר:

$$T = (1+2+1)+(1+2+1)+(1+(1+2+1)+1)+(1+2+1)+(2)+(1\cdot 6) = 26ns$$

תרגיל בית 5

שאלה 1

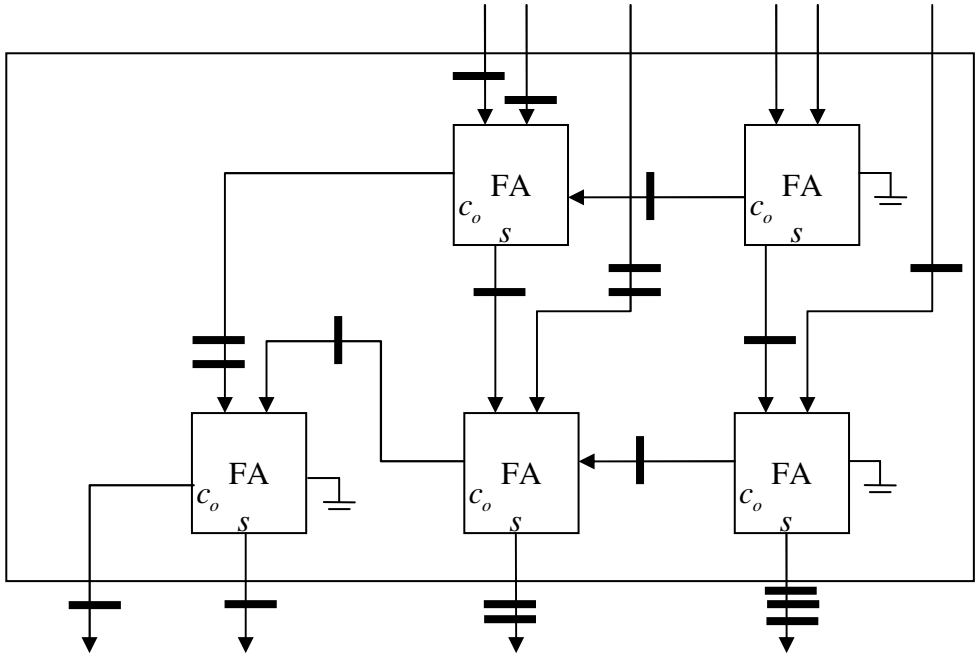
סעיף א

זמן השיהוי המירבי הוא : $T_L = 2T_{pd}(s) + 2T_{pd}(c_o) = 42ns$

ולכן התפוקה המירבית היא $TP = \frac{1}{42}$.

על מנת להגיע לתפוקה זו יש להשאיר את כניסת הרכיב יציבות למשך זמן השיהוי.

סעיף ב



מספר אוגרים: $Count(FF) = 19$

עומק הצינור: $K = 4$

תפוקה מתקבלת, תיאורטית: $TP_{\max} = \frac{1}{11}$

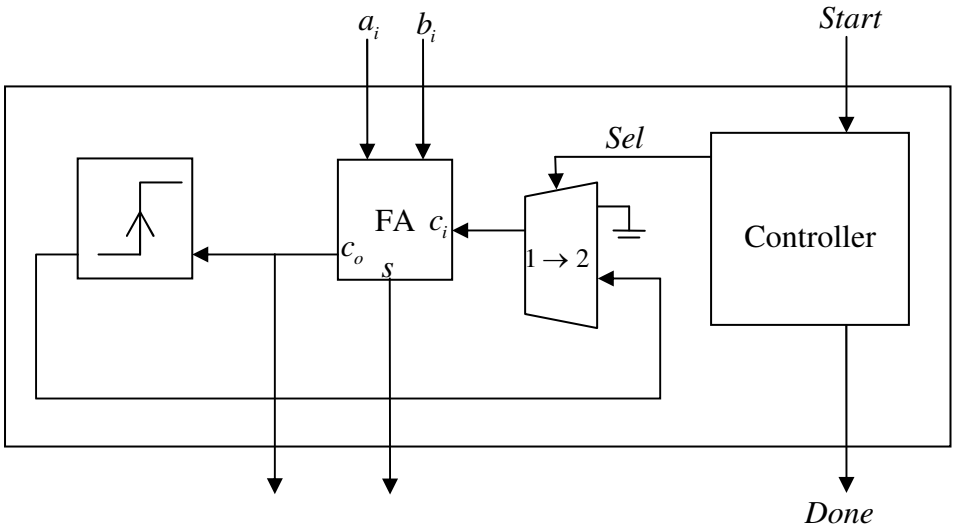
סעיף ג

נחשב ראשית את זמן המחזור המינימלי עבור הרכיב: $T_{cyc} \geq \max_{CL}\{11,10\} + T_{pd}(FF) + T_{su}(FF) = 11+5+3 = 19ns$

שיהוי: $T_L(3AdderP) = K \cdot T_{cyc} = 4 \cdot 19 = 76ns$

תפוקה מתקבלת: $TP_{\max} = \frac{1}{19}$

סעיף ד



כניסות הבקר: $Start$

יציאות הבקר: $Sel, Done$

סעיף ה

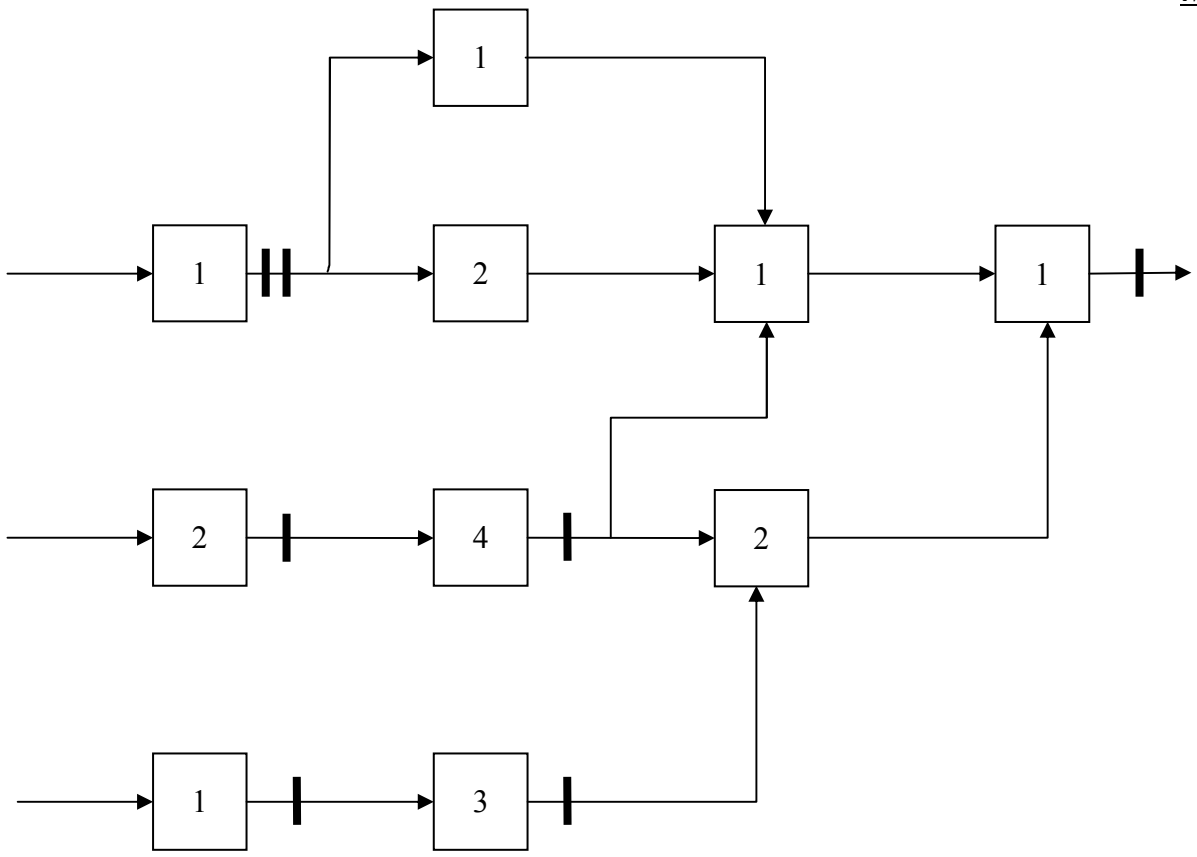
זמן המחזור המינימלי: $T_{cyc} \geq T_{pd}(FF) + T_{pd}(Sel) + T_{pd}(FA) + T_{su}(FF) = 5 + 7 + 10 + 3 = 25ns$

שיהוי המערכת, עבור שני מספרים באורך n ביטים: $T_L = n \cdot T_{cyc} = 26n$

תפוקה, עבור שני מספרים באורך n ביטים: $TP = \frac{1}{n \cdot T_{cyc}}$

שאלה 2

סעיף א



מיפוי האוגרים:
D-2, G-1, H-1, K-1, N-1,Q-1

סה"כ אוגרים: $n=7$
עומק הצינור: $K=3$
זמן מחזור מינימלי: $T_{cyc}=4$
שיהוי: $T_L=K\cdot T_{cyc}=3\cdot 4=12$

סעיף ב

Q_n	Q	n	*Latency*	# *pipe*	*TP*	T_{cyc}	מעגל
$\frac{1}{336}$	$\frac{1}{48}$	7	12	3	$\frac{1}{4}$	4	D-2, G-1, H-1, K-1, N-1,Q-1
$\frac{1}{384}$	$\frac{1}{48}$	8	12	3	$\frac{1}{4}$	4	D-1,G-1,H-1,I-1,J-1,K-1,N-1,Q-1
$\frac{1}{525}$	$\frac{1}{75}$	7	15	3	$\frac{1}{5}$	5	D-1,G-1,H-1,M-1,N-1,O-1,Q-1
$\frac{1}{648}$	$\frac{1}{108}$	6	18	3	$\frac{1}{6}$	6	D-1,G-1,H-1,O-1,P-1,Q-1
$\frac{1}{360}$	$\frac{1}{72}$	5	12	2	$\frac{1}{6}$	6	I-1,J-1,K-1,N-1,Q-1
$\frac{1}{288}$	$\frac{1}{72}$	4	12	2	$\frac{1}{6}$	6	M-1,N-1,O-1,Q-1

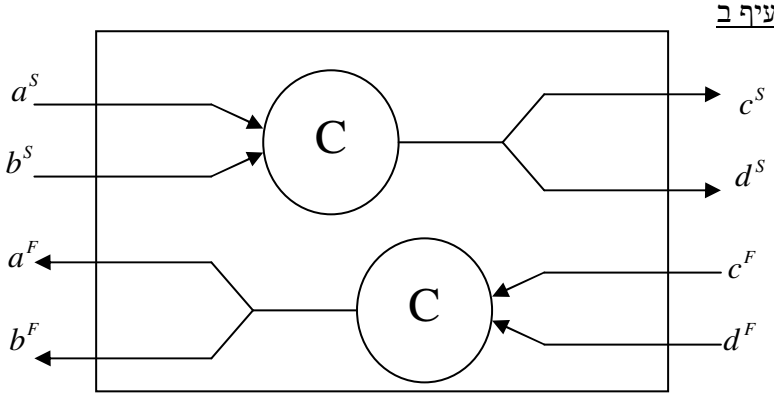
$T_{cyc}=4$ נותן את המעגל הטוב ביותר למדד TP ולמדד Q.
$T_{cyc}=6$ עם 4 אוגרים נותן את המעגל הטוב ביותר למדד Q_n.

שאלה 3

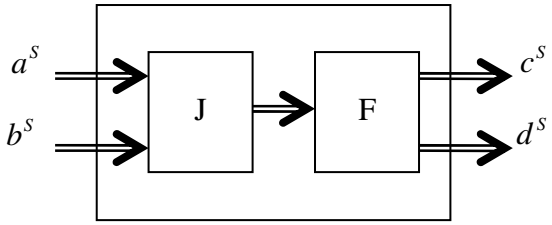
סעיף א

```
while (a OR b) do nothing;
c=0;
while (a NAND b) do nothing;
c=1;
```

סעיף ב

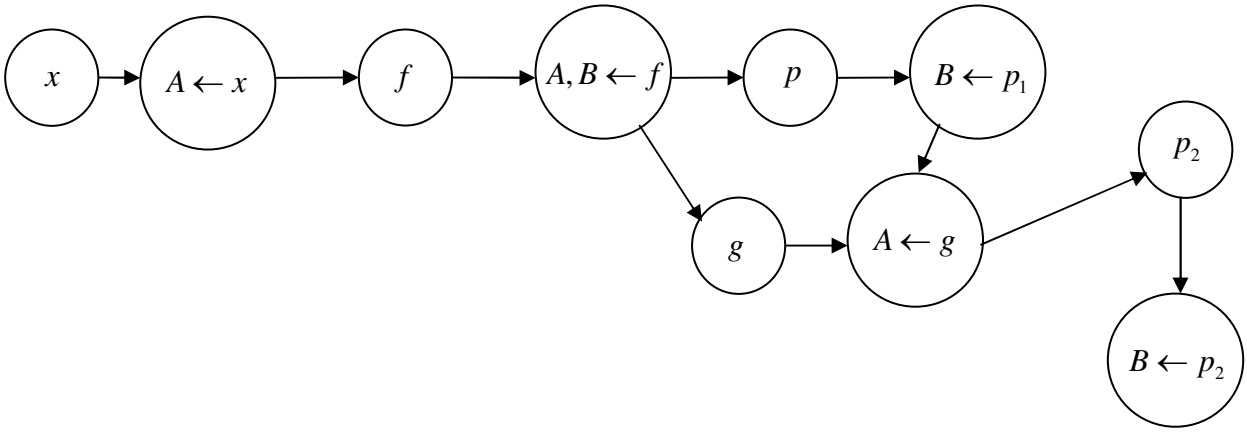


ניתן גם לממש בקר זה בעזרת בקרים קיימים:



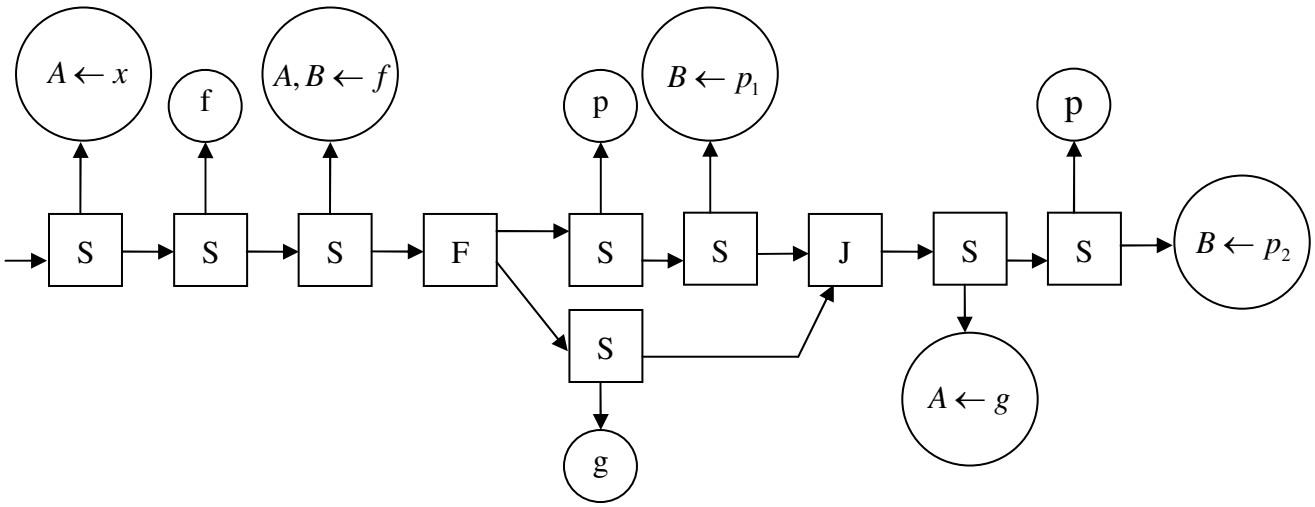
סעיף ג

יש לחשב: $t(x) = P\left(g\left(f\left(x\right)\right), P\left(f\left(x\right), f\left(x\right)\right)\right)$



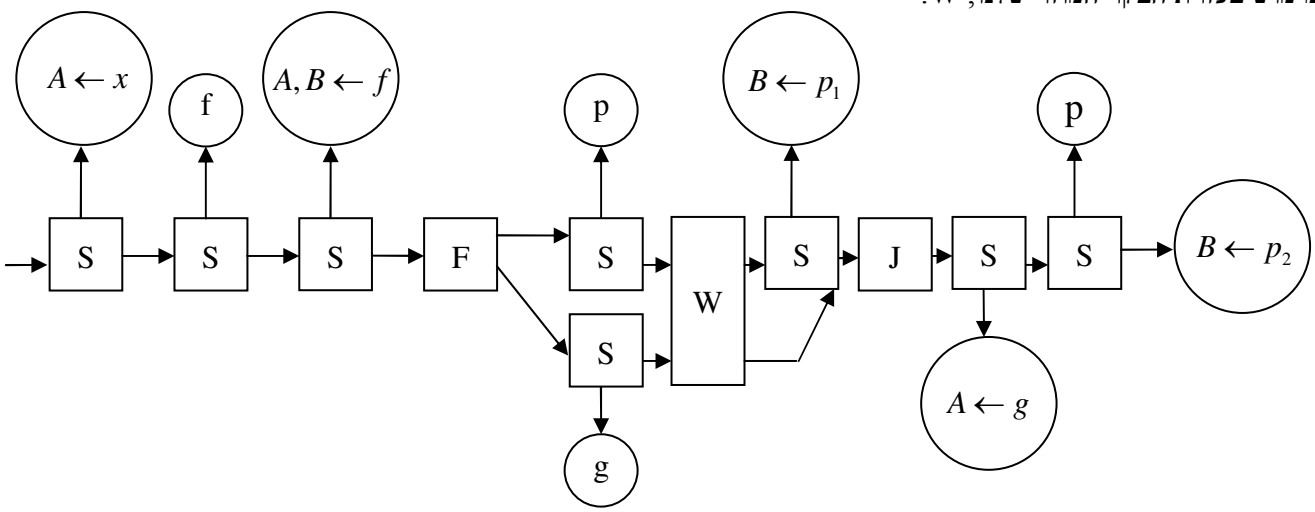
<u>סעיף ד</u>

מימוש בעזרת בקרים רגילים:



כדי להשתמש בבקר החדש שלנו, צריך לאתר חיבור של בקר Join לבקר Fork ולהכניס במקומם את הבקר החדש.

מימוש בעזרת הבקר המוזר שלנו, W:



יש לציין שמימוש זה לא חסך לנו בחומרה ולא בזמן השהייה..

לא ניתן להשתמש בבקר זה במימוש מסלול הנתונים שבעמוד 42 בגלל ריבוי הקדימויות שבו – אין שם מצב שבו שתי פעולות בוצעו ולאחרן ניתן לבצע במקביל שתי פעולות נוספות – שזו הבקרה שמספק לנו הבקר שלנו.

תרגיל בית 6

שאלה 1

סעיף א

הפקודה beqz rs,label היא פסיאודו-פקודה שקופצת ל- label אם תוכן הרגיסטר rs שווה ל- 0.
נממש פסיאודו-פקודה פקודה זו בעזרת הפקודות הבאות:
beq $rs,$zero,label

סעיף ב

הפסאודו-פקודה blt תמומש ע"י מספר פקודות אמיתיות, כאשר האחרונה מביניהן חייבת להיות פקודת branch אמיתית.
לכו נסיק שלאחר תרגום הקוד, עדיין תהיה פקודת branch (כמו בקוד המקורי) לפני שני פקודות אמיתיו אחרות של המשך הקוד. ה label יחזיק את הכתובת שיש לקפוץ אליה, יחסית לכתובת של הפקודה שאחרי ה branch.
לכן label=4.

שאלה 2

ראשית נתרגם את הפקודות לשפת אסמבלי:

000000 01111 01111 01111 00000 100000	add $t7, $t7, $t7
000100 01111 01111 1111111111111111	beq $t7, $t7, -4
001000 01111 01111 0000000000000001	addi $t7,$t7, 1

השורה הראשונה מכפילה את $t7 פי 2.
השורה השנייה היא branch שתמיד מתקיים ולכן מתבצעת קפיצה לפקודה הקודמת, יחסית לפקודה addi, כלומר הפקודה קופצת לעצמה בלולאה אינסופית – ערך $t7 לא ישתנה וישאר על 2.

שאלה 3

נתרגם את הקוד

[0x00004000]	add $t0,$zero,$zero	$t0 = 0
[0x00004004]	addi $s0,$zero,0x4004	$s0 = 0x4004
[0x00004008]	addi $t0,$t0,4	$t0 += 4
[0x0000400C]	add $s0,$s0,$t0	$s0 += $t0
[0x00004010]	jr $s0	Goto $s0

השלבים שמתבצעים:

1. איפוס של $t0 ל 0.
2. אתחול $s0=0x4004 - $s0 מצביע כרגע לפקודה שמאתחלת אותו.
3. הגדלת $t0 ב 4 ולכן כעת $t0=4.
4. הגדלת $s0 ב $t0 ולכן כעת $s0=0x4008 - $s0 מצביע כעת על הפקודה שמוסיפה 4 ל $t0.
5. פעם ראשונה שהגענו לכתובת 0x4010. קופצים ל $s0 ולכן מבצעים:
6. הגדלת $t0 ב 4 ולכן כעת $t0=8.
7. הגדלת $s0 ב $t0 ולכן כעת $s0=0x40$10 - $s0 מצביע כעת על הפקודה jr.
8. jr מבצע קפיצה לכתובת שנמצא ב $s0, וזה מצביע על אותה הפקודה, לכן נכנס ללולאה אינסופית.

בלולאה אינסופית זו, ערך האוגרים לא משתנה, וכאשר נבצע את פקודת jr בפעם החמישית, $t0 יכיל 8.

תרגיל בית 7

שאלה 1

שפת מכונה	אסמבלי	הסבר
000000 10010 00000 10010 00000 100000	add $s2, $s2, $zero	מבצע s2=s2+0, לא משנה ערכי הרגיסרטים, יכול לשמש NOP
001000 10010 10010 0000000000000000	addi $s2, $s2, 0	מבצע s2=s2+0, לא משנה ערכי הרגיסטרים, יכול לשמש NOP
000010 00000000000000000000000001	j 0x1	קופץ תמיד לפקודה השניה – יכול לשמש NOP
000100 10010 10010 0000000000000001	beq $s2, $s2 0x1	קובץ תמיד לפקודה אחת אחרי הפקודה שאחרי ה beq, כלומק לפקודה ההשלישית – לא יכול לשמש כ NOP

שאלה 2

סעיף א

כל איטרציה מכילה 8 פקודות וכותבת 2 איברים (אלא אם כן צריך רק איבר אחד – אם מספר האיברים שיש לחשב הוא אי-זוגי). לפני תחילת האיטרציות ישנן 5 פקודות. כלומר עבור יצירת סדרת פיבונצ'י באורך n נזדקק ל $9\cdot\frac{n}{2}+5=4.5n+5$ מחזורי שעון ב Single Cycle MIPS. (אם n איזוגי אז נצטרך $4.5(n-1)+8$)

סעיף ב

קוד התוכנית:

```
.data 0x10008000       # Data Segment start (assembler directive)
A: .word 0             # array element A[0] of  Fibonacci series array
.word 0                # A[1]
.word 0
.word 0
.word 0
.word 0
.word 0
.word 0
.word 0
.word 0
.word 0                # A[9]
n: .word 9             # n

.text                  # Code Segment start (assembler directive)
.globl main
main:

        la $s0, A              # load value of A into register $s0   --> $s0 = &A[0]

        la $a0, n              # load address of n into $a0  #
        lw $a0, 0($a0)         # load value of n into $a0    #  --> $a0 = n

        addi $t0, $zero, 0     # first element  $t0 = a_1 = 0
        addi $t1, $zero, 1     # second element $t1 = a_2 = 1
loop:
        sw $t0, 0($s0)         # stores element a_n
        addi $a0, $a0 -1       # decrease index: n=n-1
        beq $a0, $zero, done
        sw $t1, 4($s0)         # stores element a_n+1

        add $t0, $t0, $t1      # calculates $t0 = a_n+2
        add $t1, $t0, $t1      # calculates $t1 = a_n+3

        addi $s0, $s0, 8       # moves to the next 2 elements in the array
        addi $a0, $a0, -1      # decrease index: n=n-1
        bne $a0, $zero, loop
done:

        addi $v0, $0, 10       # exit the  program by calling syscall with parameter 10
        syscall
```

תוכן האוגרים בסוף הרצת התוכנית:

```
PC   = 00000000   EPC  = 00000000   Cause = 00000000   BadVAddr= 00000000
Status = 00000000   HI   = 00000000   LO   = 00000000
                                    General Registers
R0  (r0) = 0       R8  (t0) = 21      R16 (s0) = 268468256  R24 (t8) = 0
R1  (at) = 268435456  R9  (t1) = 34      R17 (s1) = 0       R25 (t9) = 0
R2  (v0) = 10      R10 (t2) = 0       R18 (s2) = 0       R26 (k0) = 0
R3  (v1) = 0       R11 (t3) = 0       R19 (s3) = 0       R27 (k1) = 0
R4  (a0) = 0       R12 (t4) = 0       R20 (s4) = 0       R28 (gp) = 268468224
R5  (a1) = 2147478412 R13 (t5) = 0       R21 (s5) = 0       R29 (sp) = 2147478408
```

תוכן הזיכרון בסוף הרצת התוכנית:

```
          DATA
[0x10000000]...[0x10008000]     0x00000000
[0x10008000]                        0x00000000 0x00000001 0x00000001 0x00000002
[0x10008010]                        0x00000003 0x00000005 0x00000008 0x0000000d
[0x10008020]                        0x00000015 0x00000000 0x00000009 0x00000000
[0x10008030]...[0x10040000]     0x00000000
```

שאלה 3

סעיף א

נקבל $\$v0 = 2^{n+1}$

סעיף ב

הקוד:

```
.data 0x10008000       # Data Segment start
n: .word 5             # n

.text                  # Code Segment start
.globl main
main:
        la $a0, n
        lw $a0, 0($a0)                # load $a0=n
        addi $s0, $zero, 1            # $s0 = 1
        addi $s1, $zero, 0            # $s1  = 0
loop:   slt $s2, $a0, $s1             #  if $a0 < $s1 then $s2 = 1  else $s2 = 0
        bne $s2, $zero, finish        # if $s2 == 1 goto finish
                                      # actually: if $a0 < $s1 goto finish
        add $s0, $s0, $s0             # $s0 = 2 * $s0
        addi $s1, $s1, 1              # $s1 ++
        j loop                        # goto loop
finish: add $v0, $s0, $zero           # $v0 = $s0
```

תוכן האוגרים בסוף הרצת התוכנית:

```
PC   = 00000000   EPC  = 00000000   Cause = 00000000   BadVAddr= 00000000
Status = 00000000   HI   = 00000000   LO   = 00000000
                                    General Registers
R0  (r0) = 0       R8  (t0) = 0       R16 (s0) = 64      R24 (t8) = 0
R1  (at) = 268435456  R9  (t1) = 0       R17 (s1) = 6       R25 (t9) = 0
R2  (v0) = 64      R10 (t2) = 0       R18 (s2) = 1       R26 (k0) = 0
R3  (v1) = 0       R11 (t3) = 0       R19 (s3) = 0       R27 (k1) = 0
R4  (a0) = 5       R12 (t4) = 0       R20 (s4) = 0       R28 (gp) = 268468224
R5  (a1) = 2147478412 R13 (t5) = 0       R21 (s5) = 0       R29 (sp) = 2147478408
```

תרגיל בית 8

שאלה 1

א. פקודת R-type: jr
פקודת I-type: ori
פקודת J-type: j

ב. בהנחה ששאר המחשב עובד כרגיל, החיבור יתבצע כרגיל, אך הכתיבה תתבצע לרגיסטר $s2. כלומר:
$s1=1, $s2=5, $s3=3

שאלה 2

הוספנו יחידת Zero Extend, המחוברת לביטים 0-15 של הפקודה ונכנסת דרך mux מורחב אל ה ALU.
ה ALU יבצע OR עם רגיסטר rs והנתון המורחב ל 32 ביט. לשם כך יש לתת אות ALUSrc=2, ושאר האותות מתאימים לפקודת OR רגילה.
השרטוט בדף הבא.

שאלה 3

הבעיה המרכזית כאן היא שהסתמכנו על העובדה שהקפיצות היחסיות מתבצעות יחסית ל PC+4, דבר ש"נהרס" כאן.

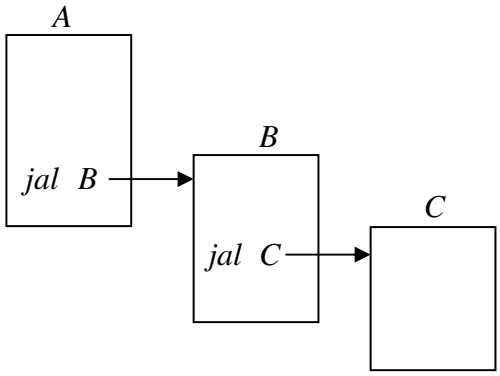
א. פקודות R-type יבוצעו בלי בעיה.
ב. פקודות lw/sw יבוצעו בלי בעיה.
ג. פקודת beq תבוצע, עם הבעיה שהוסברה לעיל – המחבר שמוסיף את כמות הפקודות שיש לקפוץ יקבל את PC ולא את PC+4 כמו שאנו רגילים ולכן הפקודה לא תקפוץ למקום שרצינו.
ד. פקודת j תבוצע, אך לא כראוי, עם אותה בעיה שבפקודת beq.

תרגיל בית 9

שאלה 1

א. השיגרה הקוראת מניחה שרגיסטרים המוגדרים Preserved on call עדיין שמישים עם ערכם שהיה לפני הקריאה לשיגרה – מבחינתה רגיסטרים אלו לא השתנו בזמן ביצוע השיגרה אליה היא קראה.
אם בכוונתה להשתמש ברגיסטרים אלו, השיגרה הנקראת חייבת לשמור את ערכם בשלב ה Prologue, ולשחזר את ערכם בשלב ה Epilogue, כדי שהשיגרה הקוראת תוכל להניח את ההנחה שלעיל.

ב. רגיסטרים $a0-$a3 מוגדרים כ Preserved on call, ולכן מבחינת השיגרה הקוראת, הם נשארים כפי שהיו לפני הקריאה לשיגרה הנקראת. כלומר, עבור השיגרה A, רגיסטר $a0 לא ישנה את ערכו וישאר 3.



איך זה קורה? בגלל ששיגרה B יודעת שהיא מתכוונת לקרוא לשיגרה C עם ארגומנט ברגיסטר $a0, היא יודעת שערכו ישתנה ולכן שומרת את ערכו במחסנית. לפני הקריאה לשיגרה C, שיגרה B תשנה את ערכו של $a0 ל 7, ובשלב ה Epilogue של שיגרה B, ישוחזר הערך של $a0 ל 3.

ג. מכיוון שרגיסטרים $v0,$v1 אינם מוגדרים כ Preseved on call, לא ניתן להניח שום הנחה לגבי ערכם של רגיסטרים אלו.
אם אנו יודעים ששיגרה B צריכה להחזיר ערכים, אז ניתן להניח שערכים אלו נמצאים ברגיסטרים אלו, כי אלו הן מוסכמות החזרת הערכים משיגרה.
כדי ששיגרה A תוכל לגשת לאותו הערך שהיה ב $v0 לפני הקריאה לשיגרה B, עליה לשמור את ערכו במחסנית, ב frame שלה, כחלק מפעולת ה pre-call, ולאחר הקריאה לשיגרה B, כחלק מפעולת ה post-call, עליה לשחזר את הערך מהמחסנית.

ד. ההיגיון הוא שאם כל הקריאות לשגרות יתבצעו תחת אותן מוסכמות, תהיה תאימות מלאה בין המתכנתים השונים, ותוכניות גדולות ומסובכות טטופלנה ע"י פירוק לחלקים קטנים, והשילוב בין החלקים הללו יהיה חלק ומסודר, אפילו אם מספר גדול של מתכנתים כתבו את החלקים השונים.
לדעתנו, ההיגיון של הגדרת הרגיסטרים הספציפיים כ Preserved on call, שנעשתה כפי שנעשה, מבוססת על סטטיסטיקה עבור השימושים ברגיסטרים בזמן קריאה לשגרה, כך שרוב הקריאות תתבצענה עם מינימום צורך לשמירה ושחזור של נתונים.

שאלה 3

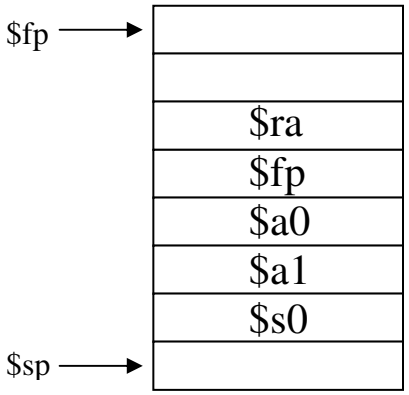
כן! כל עוד 100% מהתוכנית נכתב תחת אותה מוסכמה, התוכנית תרוץ נכון. לא משנה מי מהשגרות (קוראת או נקראת) מטפלת באילו רגיסטרים, אלא משנה שיהיה סט הגדרות אחד (מוסכמה) שיגדיר לכל רגיסטר האם הוא Preserved on call או לא, כלומר אם הוא נשמר ע"י השגרה הנקראת או לא.
בתשובתנו לשאלה זו, הנחנו כמובן שישנם גם רגיסטרים המיועדים להחזרת ערכים מהפונקציה, שהרי אם לא כך, אין טעם בשימוש בפונקציות בצורה המוכרת כיום. כלומר – חייבים להיות, תחת אותן מוסכמות, הגדרות של אוגרים למשימה זו של החזרת ערכים.

שאלה 2

א. קוד התוכנית המלא:

```
main:
      addi  $sp, $sp, -32      # Main - prologue
      sw    $ra, 20($sp)       # Main - prologue
      sw    $fp, 16($sp)       # Main - prologue
      addi  $fp, $sp, 28       # Main - prologue
      li    $a0, 12
      li    $a1, 8
      jal   gcd
      lw    $fp, 16($sp)       # Main - epilogue
      lw    $ra, 20($sp)       # Main - epilogue
      addi  $sp, $sp, 32       # Main - epilogue
      jr    $ra                # Main - epilogue
gcd:
      addi  $sp, $sp, -32      # GCD - prologue
      sw    $ra, 20($sp)       # GCD - prologue
      sw    $fp, 16($sp)       # GCD - prologue
      sw    $a0, 12($sp)       # GCD - prologue
      sw    $a1, 8($sp)        # GCD - prologue
      sw    $s0, 4($sp)        # GCD - prologue
      addi  $fp, $sp, 28       # GCD - prologue
      slt   $t0, $a0, $a1
      bne   $t0, $0, smaller
      slt   $s0, $a1, $a0
      bne   $s0, $0, greater
      addi  $v0, $a0, 0
      j     return
smaller:
      sub   $a1, $a1, $a0
      j     callagain
greater:
      sub   $a0, $a0, $a1
callagain:
      jal gcd
return:
      lw    $s0, 4($sp)        # GCD - prologue
      lw    $a1, 8($sp)        # GCD - epilogue
      lw    $a0, 12($sp)       # GCD - epilogue
      lw    $fp, 16($sp)       # GCD - epilogue
      lw    $ra, 20($sp)       # GCD - epilogue
      addi  $sp, $sp, 32       # GCD - epilogue
      jr    $ra                # GCD - epilogue
```

ב. Frame של אחד ה instances של gcd:



ג. שיגרת ה gcd האחרונה היא היחידה שתכתוב את הערך הסופי לרגיסטר $v0. מיד לאחר שתעשה זאת, היא תקפוץ לתווית return, ואז כל הקריאות שלפניה ימשיכו לתווית return, לאחר שביצעו את פקודת jal gcd שלהן. התוצאה בעצם לא תחזור אחורה כשכל עץ הקריאות מתקבל – התוצאה נכתבת בתחתית העץ, והעץ מתקפל חזרה בלי לנגוע בערך של $v0.

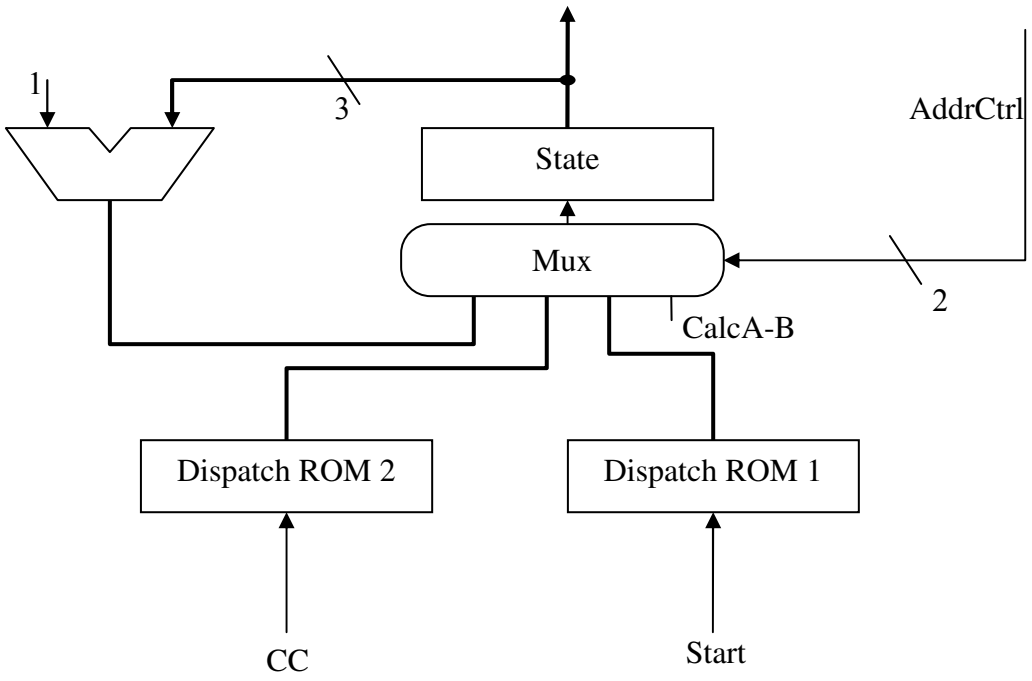
תרגיל בית 10

שאלה 1

ראשית, לא תהיה לנו בעיה עם כל פקודה של כוללת העלאת קו הבקרה MemRead.
כלומר, הבעיה תיווצר כאשר נצטרך לבצע פקודת lw – רק פקודה זו קוראת נתון מהזכרון.
אין צורך לשמור את סיביות ה opcode, כי הן יתלכלכו לנו רק בתחילת מצב 4 של הבקר – ואז כבר אין התפצלויות שדורשות שימוש ב Dispatch ROM.
כאשר מתבצע חישוב הכתובת, MemRead עדיין לא פעיל, ולכן הכל מתבצע כרגיל. בסוף ה cycle הרלוונטי, הכתובת הסופית תמתין ב ALUOut.
לאחר מכן, מתבצעת הקריאה מהזכרון והנתון נכתב לאוגר rt. מספר האוגר הזה יהרס לנו עם קריאת הנתון מהזכרון, ולכן יש לשמור אותו – סיביות [16-20].

שאלה 2

Label	dr-A	dr-B	ld-A	ld-B	dr-ALU	ALUop	Done	Sequencing
Idle					1		1	Dispatch1
LoadA	1		1					Seq
LoadB		1		1				Seq
CalcA-B						0		Dispatch2
A<-A-B			1		1	0		CalcA-B
B<-B-A				1	1	1		CalcA-B



Sequencing כולל 2 סיביות; ישנן 7 סיביות בקרה;
לבקר 6 מצבים, לכן נזדקק ל ROM בעל $2^3 = 8$ כתובות.
ולכן גודל ה ROM: $2^3 \times (7+2) = 8 \times 9 = 72bit$.

שאלה 3

מחסרת 4 מאוגר rs (ומעדכנת כך את rs) ולכתובת אליה מצביע rs (לאחר ההחסרה ב 4) נכתב תוכן אוגר rt.
כלומר:

DataMem[-4($rs)]←$rt
$rs←$rs – 4

שאלה 4