# מיתוג ומערכות סיפרתיות  תרגיל 9 – חלק א

אנא ציינו  login על התרגיל
אפשר להגיש את התרגיל בשלישיות.
עדכונים והערות יופיעו בראש האתר.

בתרגיל זה נממש ב tkgate מעגל אשר מקבל פקודות מהמשתמש ומבצע אותן (להדרכה על זיכרון, דלגלגים וקלט מהמשתמש ב  tkgate ראו בסוף התרגיל).

נכתוב 4 תוכנות קצרות על ידי השפה אשר תוגדר בחלק הבא:

1. בנו תוכנה (מאוסף הפעולות של המעבד) אשר מדפיסה מספר גדול מ 0 (וקטן מ 65000).
   כלומר התוכנה היא :
   a. Set $1 = MyNumnber
   b. Print The number on the terminal.

2. בחרו מספר בין  30000- ו 30000  וכתבו תוכנה אשר מדפיסה אותו (כולל הסימן – אם הוא שלילי).
3. כתבו תוכנה אשר קולטת מהטרמינל מספר חיובי (בין 0 ל 65000) וממירה אותו למספר בינארי. קריאת המספר נגמרת כאשר התו 0b (enter) נקרא.
4. כתבו תוכנה אשר קולטת מהטרמינל מספר (בין 30000 to 30000-) וממירה אותו למספר בינארי.
   אם המספר שלילי התו – (קוד 2D) יהיה לפני המספר. קריאת  המספר נגמרת כאשר התו 0b (enter) נקרא.

## *הפעולות של המעבד:*

על המעבד לבצע את פעולות הבאות:

| Micro instruction | Meaning | Comments |
|---|---|---|
| **Arithmetic / logic operations** | | |
| $3 = $2 + $1 | Set register $3 with $2 + $1 | |
| $3 = $2 - $1 | Set register $3 with $2 - $1 | |
| $3 = mod($2,$1) | Set register $3 with b such that $1*a + b = $2 (b < a) | Use the tkgate division |
| $3 = $2/$1 | Set register $3 with a such that $1*a + b = $2 (b < a) | Use the tkgate division |
| $3 = $2 * $1 | Set register $3 with $2 * $1 | Use tkgate multiplier |
| $3 = AND($2,$1) | Set register $3 with And($2,$1) | |
| $3 = NOT ($1) | Set register $3 with not ($1) | |
| $3 = OR($2,$1) | Set register $3 with Or($2,$1) | |
| $3 = H($1) | Set register $3 with upper 8 bits of $1 | For example (in hexadecimal) if $1 = 0f0bh then $3 = 000fh. Implement using |

| | | |
|---|---|---|
| | | the shift in the ALU |
| $3 = L($1) | Set register $3 with lower 8 bits of $1 | |
| lShift $1,$2 | Shift left $1 in $2 bits (the lower 8 bits) | |
| rShift $1,$2 | Shift right $1 in $2 bits (the lower 8 bits) | |
| Set-on-less $1,$2,$3 | $1 = 1 if $2< $3 $1=0 otherwise | |
| $1 = VAL | Put the VAL in register $1 | |
| **Memory Access operations** | | |
| Load $1,Addr($2) | Load into $1 the value in Addr+$2 | |
| Store $1,Addr($2) | Store in $1 the value in Addr+$2 | |
| Push $1 | Push $1 to the stack | |
| Push PC | Push the current value of PC to the stack. (note that the current value of PC already points to the next instruction). | |
| Pop $1 | Pop the stack into register $1. | |
| **I/O operations** | | |
| Read $1 | Read a character from the terminal into the lower 8 bits of register $1 | See the section on tty in order to see how to implement it |
| Print $1 | Print the character in the lower 8 bits of $1 | See the section on tty in order to see how to implement it |
| **Program control** | | |
| J Addr | Jump to address Addr | |
| Return $1 | Jump to address $1 | |
| Jne $1,$2,Addr | Jump to Addr if $1 is different than $2 | |
| Je $1,$2,Addr | Jump to Addr if $1 equals $2 | |
| | | |

כאשר $x הוא סימון לרגיסטר מספר x. הכוונה בסימון $3=$2+$1 היא לכך שניתן לחבר כל זוג רגיסטרים ולהציב את התוצאה ברגיסטר שלישי.

מספר הרגיסטרים של המעבד לא יעלה על 7 (רגיסטר 0 הוא 0 קבוע ולא ניתן לכתוב עליו).
מלבד רגיסטרים אלו, ניתן להשתמש ברגיסטרים אחרים (למצביע של המחסנית, ל PC , IR , MAR וכו'), אך רגיסטרים אלו לא יכללו בחישוב של הפעולות האריתמטיות.
כל הרגיסטרים הם בעלי 16 סיביות בדיוק (גם ה SP, PC וה IR) – אין להשתמש ברגיסטרים בעלי פחות או יותר סיביות.

## *אלגוריתם הפעולה של המעבד.*

המעבר עובד לפי האלגוריתם הבא:

1. CAR = 0 : טען את הפעולה הבאה לתוך IR. PC++, CAR++
2. CAR = 1: הצב ב CAR את הערך המתאים לפעולה שנמצאת בתוך ה IR.
3. CAR = ...: בצע את הפעולה. (בסופה הצב CAR=0).

שלבים 1 ו 2 נקראים שלב ה fetch .

### המחסנית

על מנת לממש קריאה לפרוצדורות אנו נזדקק למחסנית. מנגנון הקריאה עצמו יוסבר בחלק ב' של התרגיל.
על מנת לממש מחסנית אנו נחזיק רגיסטר מיוחד בשם SP. ברגיסטר זה תוחזק הכתובת בזיכרון של ראש המחסנית הנוכחי. על כן נגדיר 2 פעולות : Push ו Pop.
הפעולה Push תמומש כך:

a. push PC (or $1) - דחוף את PC למחסנית. כלומר:
   i. SP = SP+1
   ii. Load PC (or $1),SP (שים את הערך שבPC בכתובת שב SP).
b. pop $1 – הוצא את הערך בראש המחסנית והצב אותו ברגיסטר.
   i. Store $1,SP
   ii. SP = SP-1.

## *הדרכה*

1. קראו את ההדרכה על רגיסטרים, RAM ו TTY באתר של TKGate. כמו כן, מצורף באתר קובץ דוגמא לעבודה עם RAM. על מנת להבין טוב יותר את העבודה עם TTY, חברו switches לכל כניסה ו Led לכל יציאה שהיא ביט בודד, ו DIP Switch ו Led Bar לכניסות שהן 8 ביטים. זכרו שההדפסה היא בקוד ascii, ולכן, קודים שהם לא מספרים או ספרות לא יודפסו. נסו לדוגמה את 61h – האות a.
2. ממשו את המעגל על הנייר כשלב ראשון. ממשו את המעגל ב tkgate רק לאחר שממשתם את **כל** ההוראות על הנייר.
3. בנו ALU בעל הפעולות הבאות:
   c. חיבור
   d. חיסור
   e. כפל (הנכם רשאים להשתמש במעגל הכפל של tkgate).
   f. שארית בחילוק (הנכם רשאים להשתמש במעגל החילוק של tkgate).
   g. חילוק (הנכם רשאים להשתמש במעגל החילוק של tkgate).

יש לאפשר ביצוע הסטה ימינה או שמאלה של התוצאה של ה ALU במספר סיביות שמתקבל (השתמשו באיבר ההסטה של tkgate).

4. בנו טבלה של מבנה ההוראות לדוגמא:

| סוג ההוראה | תת-סוג ההוראה | אופרנד 1 | .... |
|---|---|---|---|

5. כתבו טבלה בה מתואר לכל פעולה מהו אוסף מיקרו-פעולות הנדרש כדי לממש אותה (זכרו את מיקרו הפעולה fetch).
6.
7. בנו את המעגל ללא ה control. כלומר חברו את רכיבי המערכת (לדוגמא הרגיסטרים, RAM, TTY , CAR,IR) בעזרת ה MUX וה decoder ההכרחיים.
8. מכיוון שכל פעולה יכולה להכיל כמה מיקרו פעולות, אנו זקוקים לטבלה אשר תתאם בין ההוראה לבין המיקרו הוראות.
בנו את טבלת ה CAR, כלומר לכל פעולה מהוא ה CAR הרלוונטי.

נניח להוראה אריתמטית דרושה מיקרו פעולה אחת (פרט ל fetch) ולload דרושות 2 מיקרו פעולות. אזי טבלה אפשרית היא:

| CAR | Instruction |
|---|---|
| 00010 | Arithmetic operation |
| 00011 | Load |
| 00100 | |

בטבלה זו
9. בנו טבלת אשר מתאמת בין ה CAR לcontrol :
לדוגמא,

| Input | Outputs | | |
|---|---|---|---|
| CAR | PC Load | IR load | Ect.. |
| 00000 | 1 | 1 | עוד סיביות control. |

כלומר ב CAR 00000 (דהיינו, fetch), עלינו לטעון את ה PC ואת ה IR.

מכיוון שעליכם להמיר את הטבלה לקוד הקסדצימלי מומלץ לסמן כל 4 ביטים בטבלה בצבע (או במסגרת) על מנת להאיץ את ההמרה .

10. המירו את טבלה 6 לקוד הקסדצימאלי, כתבו את קובץ הזיכרון המתאים, וחברו את ה control של המעגל.
11. כאשר נממש תוכנה נכתוב ראשית את אוסף ההוראות. לאחר מכן נארגן בטבלה אצ הערכים המתאימים ולבסוף נמיר לקוד בקסדצימאלי.
לדוגמא, התוכנה הבאה מציבה את הערכים 1 ו a0 ב $1 וב $2(בהתאמה) ומציבה ב $5 את הסכום שלהם.

| Instruction | Type (2 bits) | | Instruction (4 bits) | | | | Empty (1bit) | Operand 1 (3bits) | | | Operand 2 (3 bits) | | | Operand 3 (3 bits) | | | Value (16bits) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $1 = 0001h | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0001h |
| | 3 | | | | c | | | | 0 | | | | 1 | | | | 0001 |
| $2 = 00a0h | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 00a0h |
| | 3 | | | | c | | | | 0 | | | | 2 | | | | 00a0 |
| $5 = $2+$1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 00000h |
| | 0 | | | | | | | 0 | | | | 5 | | | | 5 | 0000h |

ולכן הקובץ הזיכרון של התוכנה יראה כך:

0/ 3c010001
1/ 3c0200a0
2/ 00550000

שימו לב מספר הביטים ביציאה הוא 32, מספר הביטים בכניסה (אי אפשר לראות זאת ישירות) הוא 16.

## Debug

מציאת שגיאות במעגל גדול אינה פשוטה. על מנת לפשט את התהליך עליכם לתת שמות עם משמעות לכניסות ולרכיבים השונים. לחיצה ממושכת על קו תוך כדי ריצה, תחשוף את ערכו בקידוד בינארי (אם הוא ביט בודד) או בהקסדצימאלי.
כך תוכלו לבדוק כל תו בקרה רלבנטי.
זכרו להתקדם בצעדים בודדים (הסימן של K עם השעון עליו).
על מנת לבדוק האם המעגל תקין לאחר השלמתו, כתבו תוכנה קצרה (בת שורה או שתיים) שבודקת כל פעולה.
לדוגמא :

1. $1 = 0f0fh
2. $2 = 10
3. Store $1,100($2)

על מנת לבדוק את תוכן הזיכרון (ועל מנת לקבוע אותו) הינכם יכולים לטעון או לכתוב אותו לקובץ. לחצו על הכפתור הימני לבחרו באופציה dump (או load).

## הערות על המימוש

1. הגודל המכסימלי של ROM הוא 32bit. אם יש צורך ביותר מ 32bit השתמשו ב2 (או יותר) ROM.
2. בגרסא 1.8.5 של tkgate צריך להוסיף buffer (שער הזהות) בין ה רגיסטר לבין כניסת הכתובת של ה RAM.
3. לאתחול המערכת הוסף switch ליציאות ה clear של הרגיסטר אשר יאתחל את כל הרגיסטרים ל 0. כלומר כל עוד ה Switch במצב off הרגיסטרים יקבעו בערך 0 . כאשר הswitch יעבור למצב on המעגל יתחיל לעבוד.
4. אין לשים ROM או RAM בתוך מודול (שימו אותו ב main, וחברו כניסה לתוך המודול).
5. הקלט נקרא ונכתב בקוד בעל 8 סיביות. אולם מכיוון שהרגיסטרים הינם בעלי 16 סיביות, נתייחס רק ל 8 סיביות התחתונות.
6. שימו לב כי סיבית ה DSR דולקת כאשר מתקבלת האות הבאה מהמשתמש. אולם אם הפעולה המתבצעת אינה קריאה של אות, סיבית הבקרה תתאפס והאות הנוכחית לא תקרא. על מנת לפתור את הבעיה הזו, בנו את הבקרה כך שקלט 0 לא יאפס את הDSR אלא ישאיר אותה בערך הנוכחי שלה. בסוף קריאת האות, אפסו את הDSR.
7. אם אין אף אות (כלומר RTS=0) קראו את התו המיוחד ff00.

8. על מנת שהכתיבה ל RAM תעבוד עלינו לבצע את השלבים הבאים:
   a. חשב את הכתובת הרצויה
   b. טען את הכתובת לאוגר הכתובת
   c. כתוב את המידע לאוגר.

שימו לב, חובה לשים שער זהות בין האוגר של הכתובת לבין כניסת ה RAM. (ראו דוגמא מצורפת).

9. כוונו את השעון להיות בעל cycle של כ 2000ns.

## מבנה קבצי הזיכרון

לקובץ זיכרון (של RAM או של ROM יש את המבנה הבא:

Address/ data data data

כלומר בזיכרון בו רוחב הכתובת הוא 4 ורוחב הפלט (כלומר המילה בזיכרון) הוא 8 הקוד הבא

a/ 0f 0a 01
e/ 01
10/ 3f

יציב במקום a את הערך 0f , במקום b את הערך 0a וכן הלאה.

קבצים אלא ניתן לטעון תוך כדי ריצה (על ידי בחירת הזיכרון ובחירת load לאחר לחיצה על הכפתור הימני), או לטעון מראש על ידי בחירת הקובץ בחלון ה properties של הזיכרון.

## הוראות הגשה.

1. יש להגיש עותק אלקטרוני של קובץ המעגל בצירוף README וכל קבצי הזיכרון הרלוונטיים. קראו לקבצי הזיכרון של תוכנות program1 program2 ו program3 1,2 ו 3.
2. יש להגיש עותק מודפס של כל המודולים אשר השתמשתם בהם לפיתרון יש להגיש

## מידע נוסף :

לקריאה נוספת פנו ל:

- Digital logic and computer design , Mano.
  (הספר באנגלית בלבד)
  - Chapter 10 Control logic design
  - Chapter 11 Computer design
  - Chapter 12 – microcomputer system design
- Computer organization and design, chapter 5 patterson and hennessy.