פרויקט Basys – דוקומנטציה

מגישים:

- אור לופטה 308256304
- עדי פולק 308552967

מסמך זה מתאר את פרויקט הBasys, על הפונקציות השונות בהן הוא משתמש. למסמך מספר חלקים:

חלק I: תיאור כללי של פעולת הכרטיס בפרויקט

מטרת הפרויקט הינה לדמות בעזרת כרטיס הBasys פעולת מעבד הקורא פקודות אסמבלי המקודדות בזיכרון – ומבצע אותן.

סימולציה זו דורשת ביצוע של מספר תהליכים במקביל.

התהליך הראשי הינו ביצוע ההוראות בזיכרון, תהליך המתבצע באמצעות לולאה בפונקציה scanArr – עליה ישנו פירוט בחלק II של המסמך.

במקביל, על התכנית לפעול בהתאם לinput שמתקבל ע"י המשתמש על גבי לוח הBasys. ניתן לחלק את input זה לשני סוגים, וכך גם הטיפול בהם מתחלק בהתאם:

(1) שימוש במתגים:
על הלוח מספר מתגים, אותם המשתמש יכול להדליק ולכבות בהתאם למידע אותו הוא מעוניין לראות על צג הLCD.
הפנייה למתגים ובדיקת המצב שלהם נעשית בשיטת הpulling. כלומר, יש מספר נקודות לאורך הקוד בהן נעשית בדיקה באמצעות תנאי If בנוגע למצב המתגים. שיטה זו מדויקת פחות (מבחינת תזמון) משיטת הInterruptים, אך זה לא משנה לנו כיוון שהשימוש בנתונים לא נעשה ע"י קטע קוד אלא ע"י עין אנושית, שלה הדיוק חשוב פחות. הקוד רץ מהר מספיק בכדי שהעין לא תבחין בכך שהשינוי מרגע הלחיצה עד להופעת המידע על המסך איטי יותר. בנוסף, שיטה זו נוחה לנו כיוון שאנחנו לא רוצים שהמידע על המעבד והזיכרון יופיע תוך כדי ביצוע פעולה. כלומר, מה שחשוב בתזמון הוא לא פרק הזמן בין הפעולות, אלא מיקום הפעולה לאורך הריצה.

(2) שימוש בכפתורים:
בניגוד למתגים, השימוש בכפתורים דורש דיוק רב יותר. הם משפיעים על ריצת הסימולטור, ובנוסף לכך לחיצה עליהם יכולה "להתפספס" אם היא לא נעשית בזמן שבו נבדקת לחיצה. לשם כך, ישנו שימוש בInterrupt כל פרק זמן של 10msec, לצורך בדיקת לחיצה. פרק זמן זה מהיר מספיק בכדי לא "לפספס" לחיצות. פירוט על השימוש בInterrupt זה נמצא בחלק II של מסמך זה.

חלק II: שימוש בInterrupts ובמשתנים גלובליים לאורך הפרויקט

בפרויקט נעשה שימוש בשני Interrupts:

Timer4

ה-Interrupt ראשון מתבסס על Timer4, שהינו Type-B Timer. משמעות הדבר היא שניתן להגדיר אותו עבור זמני מחזור ארוכים יחסית, ולכן נשתמש בו בכדי לעבוד עם רגיסטר הקלט\פלט IORegister[0]. רגיסטר זה גדל ב-1 בכל 62.5 מילישניות, ולכן נקרא לטיימר זה כל 62.5 מילישניות. הטיימר יגדיל ב-1 את הרגיסטר, ובנוסף יעלה את הexecute flag – משתנה שמטרתו להעיד האם עלינו לבצע פקודה חדשה באופן אוטומטי. כלומר, העבודה עם הדגל תיעשה רק כאשר אנחנו נמצאים במצב pause. גם מצב זה נבדק באמצעות שימוש בדגלים – פירוט על כך בסוף חלק זה.

חישוב פרק הזמן בו מתבצעת קריאה לInterrupt מתבצע באופן הבא:

תדר המעבד על הכרטיס הינו 80MHz (לאחר מעבר בPLL שמכפיל את תדר הכרטיס עצמו פי 10). תדר זה מחולק בהתאם ל-FPDIV, המקונפג להיות DIV_8. לאחר מכן, התוצאה מחולקת בהתאם לPrescaler. כיוון שהטיימר הינו Type-B, ניתן לקנפג את ערך זה להיות גבוה במיוחד באמצעות הערך TCKPS. הגדרנו את TCKPS להיות '7', כך שהתדירות מחולקת פי 256.

לאחר מכן, ניתן לחלק את המספר '1' בתדירות שקיבלנו, וכך להגיע לאורך מחזור שעון אחד. לבסוף נותן להשתמש בפרמטר PR4, שקובע כל כמה מחזורי שעון תתקבל פסיקה. הערך נבחר כך שפסיקה תתקבל כל 62.5msec, כנדרש.

Timer5

ה-Interrupt השני מתבסס על Timer5. מטרתו הינה לזהות לחיצות על כפתורים, ולכן נרצה שהקריאה לפסיקה תתבצע בקצב גבוה. בחרנו בקצב של 10msec לפסיקה.

כאמור, מטרת הפסיקה הזו הינה זיהוי לחיצה על כפתורים. לכן לא מספיק לבדוק האם כפתור הינו לחוץ, שכן לחיצה אחת אורכת מעל 10msec, ולכן היא תיספר בפועל בתור מספר לחיצות.

לשם כך, בדיקת הלחיצות נעשית בשני שלבים. בשלב הראשון, נבדקת לחיצה, ובשלב השני נבדק שחרור לחיצה. כל אחד מהם נעשה באמצעות פעולת if נפרדת ועדכון דגל המעיד האם אנו נמצאים כעת במסגרת פעולת לחיצה. כך הקריאה הראשונה שתזהה לחיצה תעלה את הדגל, וכל הקריאות הבאות תוך כדי הלחיצה לא ישפיעו על מהלך התכנית. הקריאה הראשונה שתתבצע לאחר שחרור הכפתור, תזהה שהסתיימה לחיצה, ורק אז תתבצע ההוראה עבור הכפתור שנלחץ.

בנוסף לשימוש בTimers, ישנו שימוש במספר משתנים גלובליים, שחלקם הוזכרו בתחילת חלק זה.

רשימת המשתנים הגלובליים ומטרתם

int *memArr

מצביע לזיכרון מולו הכרטיס עובד.

int regs[16]

הרגיסטרים בהם נעשה שימוש לאורך תכנית האסמבלי המקודדת.

Int IORegisters[3]

רגיסטרי החומרה.

Int pauseFlag

דגל המעיד האם הריצה הינה במצב pause.

Int executeFlag

דגל המעיד האם יש לבצע פקודה חדשה באופן אוטומטי.

Int registerToShow

מספר הרגיסטר אותו יש להציג למשתמש

int memToShow

מספר התא בזיכרון אותו יש להציג למשתמש.

int memFib[], int memLight[]

מערכים המייצגים את שתי התכניות הצרובות בכרטיס.

int BTNLflag

int BTNCflag

int BTNUflag

int BTNDflag

int BTNRflag

ארבעת הדגלים המייצגים את הכפתורים השונים, והאם הם נמצאים בתהליך של לחיצה.

חלק III: הפונקציות והקבצים השונים בפרויקט

בפרויקט ישנו שילוב בין קבצים ייעודיים לעבודה עם הכרטיס לבין קבצים שנכתבו לצורך הפרויקט.

ישנם שני קבצי c ושלושה קבצי h שנכתבו באופן ייעודי לפרויקט, ובהם חלק זה יתמקד.

הקבצים הייעודיים

main.c + sim.h

אלו הקבצים המכילים את מרבית הפונקציות שנכתבו לצורך פרויקט זה.

commands.c + commands.h

קבצים אלו מכילים את הפקודות השונות בהן הסימולטור תומך. פירוט על דרך פעולתן יימצא בהמשך חלק זה.

config_bits.h

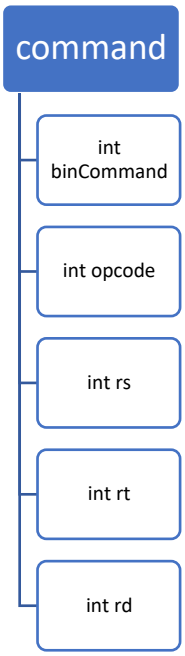
קובץ זה מכיל הגדרות קונפיגורציות שונות לכרטיס (ולשעונים בו בפרט).

מבנה הנתונים בסימולטור

Struct: command

פיענוח ההוראה אותה נרצה לבצע ייעשה אל תוך מבנה נתונים בשם command שנכתב לסימולטור.
מבנה נתונים זה מכיל חמישה משתנים מסוג int המייצגים את הפרמטרים השונים של כל פקודה:
המספר הגולמי המייצג את הפקודה במלואה (אך ללא ערך הimmediate במידה ונעשה בו שימוש),
הopcode של הפקודה, וערכי הrs, rt ו-rd של הפקודה.

עבור פירוט על הדרך בה הנתונים מחושבים מתוך ערך הפקודה הגולמי, ראה פירוט בהמשך על הפונקציות int scanArr ו- int excecute_command .



הפונקציות בסימולטור – הסבר מפורט

כעת נתאר את הפונקציות השונות בהן נעשה שימוש בסימולטור.

Branch commands

ישנן מספר פונקציות שבודקות האם תנאי מסוים מתקיים, ואם כן – מבצעות קפיצה אל כתובת בזיכרון (כולן פונקציות void, בסיום המסמך מופיע פירוט של כולן):

- beq – פונקציה המבצעת קפיצה אם ערך rs שווה לערך rt
- bgt – פונקציה המבצעת קפיצה אם ערך rs גדול מערך rt
- ble – פונקציה המבצעת קפיצה אם ערך rs קטן או שווה לערך rt
- bne – פונקציה המבצעת קפיצה אם ערך rs שונה מערך rt

קפיצה בזיכרון מתבצעת אך ורק אם הכתובת אליה קופצים קטנה מnumOfLines, אחרת התכנית תתקע (פקודת 0000). בנוסף, מתבצע בפקודה זו קידום של ערך הPC אל המקום המתאים.

Void sra(int *rd, int *rs, int *rt)

פונקציה זו מבצעת shift ימינה תוך שמירה על סימן המספר בייצוגו העשרוני. שמירת הסימן מתבצעת באופן הבא:

ראשית כל נבצע את ההזזה. לאחר מכן נחזור לרגיסטר שמכיל את הערך המקורי, לפני ההזזה, ונעתיק את הMSB שלו אל הMSB של הרגיסטר עם המספר שאחרי ההזזה, באמצעות פקודות AND ו-OR.

Void lw(int *memPtr, int *regDes)

פקודה זו טוענת מהזיכרון ערך אל תוך רגיסטר. כיוון שערכי הזיכרון מיוצגים על ידי 4 ספרות הקסדצימליות, בעוד הרגיסטר מכיל 8 ספרות כאלו, טעינת מספר שלילי דורשת פעולה מיוחדת, שכן העתקה פשוטה תגרום לכך שהמספר ברגיסטר ייקרא כמספר חיובי (הביט השמאלי ביותר ברגיסטר יהיה 0 ולא 1 כנדרש). נבחרה השיטה הבאה:

ראשית, מתבצעת בדיקה האם המספר בזיכרון הינו שלילי על ידי פעולת AND עם מספר שמכיל '1' בביט מס' 15 ו'0' בכל האחרים. בצורה זו נדע האם המספר אותו אנו טוענים אמור להיות שלילי. אם הוא אינו שלילי, הטעינה התבצעה כראוי. אם לא, נהפוך את כל 16 הביטים השמאליים ל'1' על ידי פעולת OR עם המספר ההקסדצימלי 0xFFFF0000.

Void sw(int *memPtr, int *regDes)

פונקציה זו שומרת בזיכרון ערך הנמצא באחד הרגיסטרים. כיוון שהייצוג בזיכרון הינו ע"י 16 סיביות (ולא 32 כמו ברגיסטרים), נאפס את 16 הביטים השמאליים ע"י פעולת AND עם 0xFFFF.

Void lhi(int *regDes, int *regSrc)

פונקציה זו מכניסה לחצי השמאלי של ערך רגיסטר כלשהו rd את החצי הימני של רגיסטר כלשהו rs. לשם כך, נלקח ערך הרגיסטר rs, מתבצעת הזזה שלו שמאלה ב16 ביטים (כדי להעביר את החצי הימני שלו לחצי השמאלי של ערך הרגיסטר המעודכן), ולבסוף חיבור של החצי הימני של רגיסטר rd המקורי. כך נקבל את rd המעודכן.

Void newOp(int *rd, int *rs, int *rt)

זוהי פקודה שמטרתה לתמוך בקלט\פלט באמצעות רגיסטרי החומרה. אם ערכו של rt הינו 0, הפקודה כותבת לרגיסטר rd את הערך שברגיסטר החומרה rs. אחרת, היא כותבת לרגיסטר החומרה rs את ערכו של הרגיסטר rd.

Void memToArr()

פונקציה זו בודקת את מצב המתג SW7, ובהתאם לו מעבירה את המצביע memArr אל תמונת הזיכרון מולה נעבוד בפרויקט. בהתאם לדרישה, הקריאה לפונקציה זו נעשית רק פעם אחת, עם עליית הכרטיס.

Void changeLed(int *IORegisters)

פונקציה זו מקבלת מצביע לרגיסטרי החומרה, ופונה אל הרגיסטר בו מתואר מצב הנורות הרצוי. היא עוברת על הערכים בו, ומכבה\מדליקה נורות LED לפי הדרוש.

Int excecute_command(struct command newComm, int *regs, int *IOregisteres, int *pc)

הפונקציה מקבלת struct מסוג command, ובודקת מה הפקודה שצריכה להתבצע באמצעות רצף תנאים על ערך הopcode של הפקודה. לסיום הפונקציה תחזיר '1' בכדי להמשיך לקריאת הפקודה הבאה בscanArr (הפונקציה שקראה לה), אלא אם כן הפקודה שבוצעה הינה פקודת halt שעוצרת את התכנית – ואז היא תחזיר '0'.

void updateComm(struct command *newComm, int *pc)

הפונקציה מקבלת את ערך הpc בו נמצאת הפקודה הבאה שצריכה להתבצע, ומעדכנת את הרכיבים השונים של הstruct באמצעות פעולות AND (לאיפוס החלקים בפקודה הגולמית שלא מעניינים אותנו) ו-shift.

void testImm(struct command newComm, int *pc)

הפונקציה מקבלת את ערך הpc בו נמצאת הפקודה הבאה שצריכה להתבצע, ואת הפקודה עצמה. היא בודקת האם הפקודה כוללת שימוש בimmediate. אם כן, נעדכן את ערך הPC וניקח מהזיכרון את המספר שמייצג את ערך הimmediate. לאחר מכן, נבדוק האם הוא מייצג מספר שלילי. אם כן, נהפוך את 16 הביטים השמאליים שלו ל'1', שכן המספר מיוצג בזיכרון ע"י 16 ביטים אך לאחר הטעינה הוא מיוצג ע"י 32, ונרצה לשמור עליו שלילי. ערך הImmediate נשמר ברגיסטר מספר 1. אם אין שימוש בImmediate, נשמור ברגיסטר זה '0'.

void displayOnScreen(int *lastSWState, char *strToLCD, int *count, int *firstMemSW, struct command *newComm)

מטרת הפונקציה להציג על מסך הLCD את הערך שהמשתמש רוצה לראות. לשם כך נבדקים מצבי המתגים הרלוונטיים, והמידע הרלוונטי מודפס למסך באמצעות פקודות sprintf ו-LCD_WriteStringAtPos.

נקודה חשובה בנוגע לשימוש בפונקציה היא השימוש בפקודת LCD_DisplayClear(). פקודה זו מוחקת את המידע שעל תצוגת הLCD, אך לא נוכל להשתמש בה לפני כל כתיבה, שכן התוצאה תהיה ריצוד של המסך. לשם כך אנו משתמשים במשתנה lastSWState. הסיבה היא שכל עוד המתגים נמצאים באותו המצב, אין צורך לנקות את המסך – שכן הפלט יהיה באותו האורך ופשוט ידרוס את הפלט הקודם. אם מצב המתגים כן ישתנה – יכול להיות שאורך הפלט החדש יהיה קצר מאורך הפלט הקודם, ולכן נרצה לבצע פעולת מחיקה לתגובה שעל המסך לפני הכתיבה החדשה.

Int scanArr(int *pc, int *regs, , int *IORegisters)

פונקציה זו אחראית על סריקת מערך הזיכרון. היא מתחילה את הסריקה מכתובת 0, אותו היא מחזיקה במשתנה ה-pc, ורצה בלולאה לאחר אתחול הפעולה הראשונה שיש לבצע. כל איטרציה של הלולאה מתחילה בהצגת המידע שהמשתמש מעוניין בו על גבי תצוגת הLCD, ולאחר מכן בדיקה האם יש לבצע פעולה חדשה. פעולה חדשה תתבצע בעליית דגל הexecute flag.

לאחר מכן ביצוע פקודה מתבצע קידום של ערכי count ו-pc. בנוסף, דגל הexecute יורד ומתבצעת בדיקה האם התבצעה כעת פקודת halt. אם לא – טוענים את הפקודה הבאה. אם כן – יוצאים מהלולאה ונכנסים ללולאה אינסופית המבצעת כתיבה למסך בהתאם למצב המתגים.

Int main(int argc, char *argv[])

מטרת פונקציה זו היא לדאוג לאתחול הרכיבים השונים על הכרטיס והאינטרקציה מולם, בנוסף לקריאה לפונקציות שמכינות את הInterrupts בתכנית.

לאחר מכן ישנה קריאה לscanArr שמכילה לולאה אינסופית, כך שישנה חזרה לmain רק במקרה של תקלה.

רשימת הפונקציות המלאה בסימולטור

```
void add(int *rd, int *rs, int *rt);

void sub(int *rd, int *rs, int *rt);

void andCom(int *rd, int *rs, int *rt);

void orCom (int *rd, int *rs, int *rt);

void sll(int *rd, int *rs, int *rt);

void sra(int *rd, int *rs, int *rt);

void newOp(int *rd, int *rs, int *rt);

void beq(int *rd, int *rs, int *rt, int *pc);

void bgt(int *rd, int *rs, int *rt, int *pc);

void ble(int *rd, int *rs, int *rt, int *pc);

void bne(int *rd, int *rs, int *rt, int *pc);

void jal(int *rd, int *pc, int *r15);
```

```
void lw(int *memPtr, int *regDes);

void sw(int *memPtr, int *regDes);

void lhi(int *regDes, int *regSrc);

void load_Fib();

void load_Lights();

void memToArr();

void changeLed(int *IORegisters);

int excecute_command(struct command newComm, int *regs, int *IORegisters, int
*pc);

void testImm(struct command newComm, int *pc);

void updateComm(struct command *newComm, int *pc);

void newCommFunc(struct command *newComm, int *pc);

void initComm(struct command *newComm, int *pc);

void displayOnScreen(int *lastSWState, char *strToLCD, int *count, int *firstMemSW,
struct command newComm);

void scanArr(int *pc, int *regs, int *IORegisters);

void Timer5Setup();
```

```
void Timer4Setup();

void __ISR(_TIMER_5_VECTOR, ipl2) Timer5ISR(void);

void __ISR(_TIMER_4_VECTOR, ipl2auto) Timer4ISR(void);

int main (int argc, char** argv);
```

חלק IV: תכנית הנורות

לאורך תוכנית הנורות נעשה שימוש במספר רגיסטרים:

רגיסטר	תפקיד
$s2	הכיוון בו נורות הLED נדלקות. עבור $s2=0 הכיוון הוא שמאל, אחרת - ימין
$v0	הערך שלפיו ידלקו הנורות. לדוגמא, עבור הדלקת הנורה הימנית ביותר, $v0=1=0b1, עבור השני מימין $v0=2=0b10, וכו'.
$t1, $s1	$s1 – מחזיק את מספר הלחיצות העדכני על הכפתור BTNC. $t1 – מספר הלחיצות על הכפתור לפני תחילת ההמתנה של 5 שניות.
$t0, $s0	$t0- מונה הזמן מאז הreset האחרון. $s0- הערך העתידי הרצוי במונה בעת שינוי מצב הנורות.
$a0, $a1	קבועים המגדירים את נקודות הקיצון של מצבי $v0. $a0=1=0b00000001 - הנורה הראשונה מימין $a1=128=0b10000000 - הנורה השמינית (והאחרונה) מימין.

דרך פעולת התוכנית:

התוכנית מתחילה בהדלקת הנורה הראשונה, נורה מספר אחת (IORegister[1]= 00000001 כאשר נתעלם מהביטים השמאליים לבינתיים כפי שנאמר ונתייחס אליהם כשיהיו רלוונטיים). ואיפוס הכיוון לשמאל, ($s2=0). כיוון שתנועת הנורה הבאה חייבת להיות לכיוון שמאל, התכנית תקפוץ לlabel המתאים לתנועה וודאית שמאלה (LEFTANYWAY).

בדומה, התכנית בודקת בכל שלב בריצה:

- אם הנורה הראשונה דולקת- הכיוון חייב להיות שמאל, ללא קשר במספר הלחיצות.

- אם הנורה האחרונה דולקת- הכיוון הוא ימין.

במידה ואף אחד מהתנאים לא התקיים-

לאחר כל שינוי של מצב הנורות, אנו שומרים ברגיסטר $s1 את מספר הלחיצות בכפתור BTNC. לאחר המתנה של 5 שניות (פירוט בהמשך) נרצה לקדם את הנורה. לכן נרצה לדעת אם יש לשנות את כיוון ההתקדמות המקורי. בכדי לעשות זאת, נטען את מספר הלחיצות העדכני ל$s1, אך לפני כן נשמור את ערך $s1 הישן ברגיסטר $t1. באמצעות חיסור ביניהם וביצוע פעולת AND בין ערכי הרגיסטרים, נדע האם יש לשנות את הכיוון (שכן שינוי הכיוון יתבצע עבור הפרש אי-זוגי). במידה ויצא 0 – מספר זוגי של לחיצות- יש לשמור על הכיוון שהיינו בו. במידה ו1- מספר א"ז- יש להתקדם בכיוון הנגדי.

אם כיוון ההתקדמות הינו שמאלה - נשתמש ב sll ונכפול את המספר ב2 על מנת שבהצגה הבינארית ה1 יתקדם שמאלה. אם הכיוון הוא ימין, נבצע sra- חילוק ב2 והזזת הביט הדולק לכיוון ימין.

המתנת 5 שניות בין הדלקת נורות:

לאורך התכנית ישנו שימוש ברגיסטר $s0 שמכיל את הערך העתידי הרצוי במונה IORegisters[0] בעת השינוי הבא במצב הנורות. לכן, לפני כל שינוי שנדרש במצב הנורות, נטען את הזמן שעבר מה reset האחרון למשתנה $t0, נחסר את הזמן לפני הדלקת הנורה ששמרנו ונבדוק האם הזמן הנוכחי גדול מהזמן הדרוש. במידה ולא- נטען שוב את הזמן ונבדוק שוב את החיסור עד שיעברו 5 שניות.

רשימת הLabels

בתכנית האמסבלי ישנו שימוש במספר Labels האחראיים על הכוונת שינוי הLED-ים לכיוון המתאים.

הLabel בו נתחיל (לאחר אתחול התכנית) הוא LEFTANYWAY וממנו נתקדם בהתאם.

שם הLabel	תפקיד הLabel	כיווני התקדמות אפשריים
MOVELEFT	הזזת הנורה הדולקת שמאלה ושמירת מספר הלחיצות על BTNC לאחר התזוזה.	WAIT5SEC RIGHTANYWAY
RIGHTANYWAY	המתנה של 5 שניות ולאחריה הכנה לתזוזה ימינה.	MOVERIGHT
MOVERIGHT	הזזת הנורה הדלוקת ימינה ושמירת מספר הלחיצות על BTNC לאחר התזוזה.	WAIT5SEC LEFTANYWAY
LEFTANYWAY	המתנה של 5 שניות ולאחריה הכנה לתזוזה שמאלה	MOVELEFT
WAIT5SEC	המתנה של 5 שניות ובחירת כיוון ההתקדמות בהתאם למספר הלחיצות על BTNC משינוי מצב הLED-ים האחרון ועד לסוף הטיימר שתחת Label זה.	LASTLABEL CHANGETORIGHT MOVELEFT
CHANGETORIGHT	משנה את כיוון התנועה לימין (רגיסטר $s2)	MOVERIGHT
LASTLABEL	מכוון לתזוזה שמאלה או ימינה בהתאם לדרוש	MOVELEFT MOVERIGHT