בס"ד

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

קובץ הכנה ניסוי מעבדה מס' 1 – חלק 3

Tutorial 1.3 – Instructions Coding and Timing

מעבדת מיקרומחשבים – המחלקה להנדסת חשמל ומחשבים

מס' קורס - 361.1.3353

כתיבה ועריכה: חנן ריבוא

מהדורה 1 – שנה"ל תשע"ו

A. הקדמה:

במסמך זה נדון בלימוד צורת הקידוד של סט פקודות הליבה (פקודות אסמבלי) של מיקרו-בקר MSP430. בנוסף, נלמד מהו משך זמן לביצוע כל פקודה (ב- cycles) וגודל אחסון כל פקודה בזיכרון (ב- word, במקרה שלנו 16bit).

בשונה מארכיטקטורת SRC, המשמשת כארכיטקטורה אבסטרקטית לצורך לימוד. ארכיטקטורת MSP430, הינה מעשית ומשמשת לעבודתכם במעבדה וללימוד יישומי של החומר התיאורטי.

ברצוני לחדד מס' נקודות לפני שנמשיך הלאה:

- ★ אסמבלר - תוכנה שנמצא בסביבת הפיתוח וממירה קוד בשפת אסמבלי לקוד בשפת מכונה (מותאם לשפת מכונה ספציפית – במקרה שלנו בקר MSP430).
- ★ כמו שלמדנו, בשפת אסמבלי (של בקר MSP430) יש 27 **פקודות ליבה** (Core Instruction) שרק הן מתורגמות לקוד מכונה ע"י האסמבלר ועוד 24 **פקודות חיקוי** (Emulated Instruction) הנועדות לנוחות המשתמש ומומרות ישירות לפקודות ליבה ע"י האסמבלר ורק אז מקודדות לקוד מכונה.
- ★ בשונה מפקודות (Instructions) ישנן הוראות (**Directives**) שאין להם תרגום לקוד מכונה (אינן רצות ב-CPU). הוראות אלו נועדו לאסמבלר לצורך ביצוע פעולות מקדימות, למשל: הקצאת מקום בזיכרון עבור משתנים/מחרוזות, תיחול ערכי משתנים, חלוקת הזיכרון למקטעים מסוימים וכו'.
- ★ ישנם 2 סוגי זיכרונות:
  - ✓ **זיכרון RAM** - זיכרון זה נדיף (כאשר **מכבים** את המתח מהבקר תוכנו נמחק). כאשר מבצעים RESET תוכן זיכרון ה-RAM נשמר (ביצוע RESET משמעו טעינת ערך כתובת הפקודה הראשונה של התוכנית לרגיסטר PC **ולא איפוס מתח הבקר**).
    זיכרון זה נועד לשימוש "שולחן עבודה- דף טיוטה" עבור התוכנית שלנו ולשימוש המחסנית (STACK).
  - ✓ **זיכרון FLASH** - זיכרון זה אינו נדיף (כאשר מכבים את המתח מהבקר תוכנו נשמר).
    סביבת הפיתוח טוענת לאזור זה בזיכרון את תוכנית הקוד שנכתוב ואת הקבועים אותם נרצה לשמור ללא תלות במצב המתח של הבקר.
    כדי לבצע **כתיבה לזיכרון FLASH** מתוך תוכנית שנכתוב לא נוכל לכתוב ישירות לזיכרון זה.
    יש צורך לעבוד עם מתווך הנקרא Flash Controller (נלמד זאת בקורס "מבנה מחשבים ספרתיים"). **לעומת זאת קריאה מזיכרון זה נבצע כרגיל.**
    בקורס זה, סביבת הפיתוח תעשה את הכתיבה ל-FLASH בשבילנו בשלב מקדים לפני טעינת הקוד.
    ה-Directives המתאימים לצורך כך,

```
RSEG    CODE
; your code
END
```

```
RSEG    DATA
; your constant
END
```

שימוש ב- DATA **RSEG** מצריך שינוי קטן בקובץ ה- Linker של ברירת המחדל.

כדי להימנע מזה, אנו נעדכן משתנים קבועים **בזיכרון ה-FLASH** בצורה הבאה:

בתחילת זיכרון ה-FLASH ימוקמו **הקבועים** ואחריהם תמוקם **התוכנית**.

```
RSEG    CODE
; your constant
; your code
END
```

B. תהליך שליפה פיענוח וביצוע של פקודת אסמבלי:

- כפי שלמדתם בקורס "מבוא למחשבים", לאחר שתוכנית נצרבת לזיכרון הבקר היא מוכנה לביצוע. לאחר פעולת RESET כתובת ה- byte הראשון של הפקודה הראשונה מועבר לרגיסטר PC ותוכן הפקודה מועבר לרגיסטר IR. ערך רגיסטר PC מתעדכן בכתובת של הפקודה הבאה לביצוע.
- תהליך ביצוע פקודה מתבצע בשני שלבים:
  1. שלב השליפה (Fetch) – הבאת קוד הפקודה מהזיכרון לרגיסטר IR.
  2. שלב הפיענוח (Decoding) והביצוע (Executing) של הפקודה.
- במשך זמן ביצוע פקודה בודדת מתבצעת הפקודה הנוכחית וגם מתבצעת שליפה של הפקודה הבאה (pipeline).

C. קידוד פקודות הליבה – חלוקה לשלושה פורמטים שונים (שלוש קבוצות של פקודות):

27 פקודות הליבה מחולקות לשלוש קבוצות של פקודות: **Double** operand ,**Single** operand, **jmp**

לדוגמה: **mov**(.b) src,dst | **inc**(.b) src | **jmp** label

כיצד החומרה יודעת באיזה פורמט כל פקודת אסמבלי ? לפי ערך של 4-bit MSB (ביטים 12-15)

1. פקודות Double operand – ישנן 12 פקודות: ערך של 4-bit MSB בין 0x4-0xF

חלוקת הקידוד לשדות

15 14 13 12	11 10 9 8	7	6	5 4	3 2 1 0
Op-code	S-Reg	Ad	B/W	As	D-Reg

Instruction	Opc - HEX
MOV(.B) src,dst **src → dst**	4
ADD(.B) src,dst **src + dst → dst**	5
ADDC(.B) src,dst **src + dst + C → dst**	6
SUB(.B) src,dst **dst + .not.src + 1 → dst**	8
SUBC(.B) src,dst **dst + .not.src + C → dst**	7
CMP(.B) src,dst **Dst - src**	9
DADD(.B) src,dst **src + dst + C → dst (decimally)**	A
BIT(.B) src,dst **src .and. dst**	B
BIC(.B) src,dst **.not.src .and. dst →dst**	C
BIS(.B) src,dst **src .or. dst →dst**	D
XOR(.B) src,dst **src .xor. dst → dst**	E
AND(.B) src,dst **src .and. dst → dst**	F

הערה: שדה OPC בגודל 4bit עבור 12 פקודות.

קודים 1,2,3 לא כלולים וזאת כדי להפנות ל-2 הפורמטים הנוספים (jump , single operand)

טבלה המכילה דוגמא עבור פקודת mov בשילוב כל שיטות המען

Examples	Machine Code - HEX	Opc	S-Reg	Ad	B/W	As	D-Reg	Cycles	Length(W)
Mov R5,R4	4504	**0100**	0101	**0**	0	**00**	0100	1	1
Mov R5,6(R4)	4584 0006	**0100**	0101	**1**	0	**00**	0100	4	2
Mov R5,EDE	4580 abs.add(EDE)-(PC+2)	**0100**	0101	**1**	0	**00**	0000	4	2
Mov R5,&EDE	4582 abs.add(EDE)	**0100**	0101	**1**	0	**00**	0010	4	2
Mov @R5,R4	4524	**0100**	0101	**0**	0	**10**	0100	2	1
Mov @R5,6(R4)	45A4 0006	**0100**	0101	**1**	0	**10**	0100	5	2
Mov @R5,EDE	45A0 abs.add(EDE)–(PC+2)	**0100**	0101	**1**	0	**10**	0000	5	2
Mov @R5,&EDE	45A2 abs.add(EDE)	**0100**	0101	**1**	0	**10**	0010	5	2
Mov @R5+,R4	4534	**0100**	0101	**0**	0	**11**	0100	2	1
Mov @R5+,6(R4)	45B4 0006	**0100**	0101	**1**	0	**11**	0100	5	2
Mov @R5+,EDE	45B0 abs.add(EDE)–(PC+2)	**0100**	0101	**1**	0	**11**	0000	5	2
Mov @R5+,&EDE	45B2 abs.add(EDE)	**0100**	0101	**1**	0	**11**	0010	5	2
Mov #0x40,R4	4034 0040	**0100**	0000	**0**	0	**11**	0100	2	2
Mov #0x40,6(R4)	40B4 0040 0006	**0100**	0000	**1**	0	**11**	0100	5	3
Mov #4,8(R4)	42A4 0008	**0100**	0010	**1**	0	**10**	0100	5	3
Mov #0x40,EDE	40B0 0040 abs.add(EDE)–(PC+4)	**0100**	0000	**1**	0	**11**	0000	5	3
Mov #0x40,&EDE	40B2 0040 abs.add(EDE)	**0100**	0000	**1**	0	**11**	0010	5	3
Mov 6(R5),R4	4514 0006	**0100**	0101	**0**	0	**01**	0100	3	2
Mov 6(R5),6(R4)	4594 0006 0006	**0100**	0101	**1**	0	**01**	0100	6	3
Mov 6(R5),EDE	4590 0006 abs.add(EDE)–(PC+4)	**0100**	0101	**1**	0	**01**	0000	6	3
Mov 6(R5),&EDE	4592 0006 abs.add(EDE)	**0100**	0101	**1**	0	**01**	0010	6	3
Mov TONI,R4	4014 abs.add(TONI)–(PC+2)	**0100**	0000	**0**	0	**01**	0100	3	2
Mov TONI,6(R4)	4094 abs.add(TONI)–(PC+2) 0006	**0100**	0000	**1**	0	**01**	0100	6	3
Mov TONI,EDE	4090 abs.add(TONI)–(PC+2) abs.add(EDE)–(PC+4)	**0100**	0000	**1**	0	**01**	0000	6	3
Mov TONI,&EDE	4092 abs.add(TONI)–(PC+2) abs.add(EDE)	**0100**	0000	**1**	0	**01**	0010	6	3
Mov &TONI,R4	4214 abs.add(TONI)	**0100**	0010	**0**	0	**01**	0100	3	2
Mov &TONI,6(R4)	4294 abs.add(TONI) 0006	**0100**	0010	**1**	0	**01**	0100	6	3
Mov &TONI,EDE	4290 abs.add(TONI) abs.add(EDE)–(PC+4)	**0100**	0010	**1**	0	**01**	0000	6	3
Mov &TONI,&EDE	4292 abs.add(TONI) abs.add(EDE)	**0100**	0010	**1**	0	**01**	0010	6	3

הערות חשובות:

a. **הטבלה הנ"ל נכונה עבור כל פקודות double operand** למעט ערך שדה op-code (השונה בין הפקודות).

b. פקודה המכילה קבוע מתוך אלו שבטבלה הבאה. במקרה זה, הקבוע מיוצר ע"י רגיסטרים R3,R2 ולכן אינו מאוחסן בזיכרון לאחר קידוד הפקודה. במקרה זה, ערך **S-Reg** יהיה שווה 2 **או** 3 כמצוין בטבלה. במקרה שישנם 2 קבועים המופיעים בטבלה. הקבוע הראשון ייוצר ע"י R3,R2 והקבוע השני יאוחסן בזיכרון (ראה שורה מסומנת בצהוב בטבלה הנ"ל).

Register	As	Constant	Remarks
R2	00	- - - - -	Register mode
R2	01	(0)	Absolute address mode
R2	10	00004h	+4, bit processing
R2	11	00008h	+8, bit processing
R3	00	00000h	0, word processing
R3	01	00001h	+1
R3	10	00002h	+2, bit processing
R3	11	0FFFFh	-1, word processing

2. פקודות Single operand – ישנן 7 פקודות: ערך של 4-bit MSB שווה 0x1

חלוקת הקידוד לשדות

15 14 13 12 11 10 9 8 7	6	5 4	3 2 1 0
Op-code	B/W	Ad	D/S-Reg

Instruction	Opc - Binary
RRC(.B) dst **C → MSB →.....LSB → C**	**0001** 0000 0
RRA(.B) dst **MSB → MSB →....LSB →C**	**0001** 0001 0
PUSH(.B) src **SP -- 2 → SP, src →@SP**	**0001** 0010 0
SWPB dst **Swap bytes**	**0001** 0000 1
CALL dst **SP -- 2 → SP, PC+2 →@SP** **dst → PC**	**0001** 0010 1
RETI **TOS → SR, SP + 2 →SP** **TOS → PC,SP + 2 → SP**	**0001** 0011 0
SXT dst **Bit 7 → Bit 8.....Bit 15**	**0001** 0001 1

טבלה המכילה דוגמא לכל אחת מהפקודות לכל שיטות המען

Examples	Machine Code - HEX	Opc	B/W	Ad	D/S-Reg	Cycles	Length(W)
RRC R5	1005	**0001** 0000 0	0	**00**	0101	1	1
RRA R5	1105	**0001** 0001 0	0	**00**	0101	1	1
PUSH R5	1205	**0001** 0010 0	0	**00**	0101	3	1
SWPB R5	1085	**0001** 0000 1	0	**00**	0101	1	1
CALL R5	1285	**0001** 0010 1	0	**00**	0101	4	1
SXT R5	1185	**0001** 0001 1	0	**00**	0101	1	1
RRC @R5	1025	**0001** 0000 0	0	**10**	0101	3	1
RRA @R5	1125	**0001** 0001 0	0	**10**	0101	3	1
PUSH @R5	1225	**0001** 0010 0	0	**10**	0101	4	1
SWPB @R5	10A5	**0001** 0000 1	0	**10**	0101	3	1
CALL @R5	12A5	**0001** 0010 1	0	**10**	0101	4	1
SXT @R5	11A5	**0001** 0001 1	0	**10**	0101	3	1
RRC @R5+	1035	**0001** 0000 0	0	**11**	0101	3	1
RRA @R5+	1135	**0001** 0001 0	0	**11**	0101	3	1
PUSH @R5+	1235	**0001** 0010 0	0	**11**	0101	5	1
SWPB @R5+	10B5	**0001** 0000 1	0	**11**	0101	3	1
CALL @R5+	12B5	**0001** 0010 1	0	**11**	0101	5	1
SXT @R5+	11B5	**0001** 0001 1	0	**11**	0101	3	1
PUSH #0x40	1230 0040	**0001** 0010 0	0	**11**	0000	4	2
CALL #0x3200	12B0 3200	**0001** 0010 1	0	**11**	0000	5	2
RRC 6(R5)	1015 0006	**0001** 0000 0	0	**01**	0101	4	2
RRA 6(R5)	1115 0006	**0001** 0001 0	0	**01**	0101	4	2
PUSH 6(R5)	1215 0006	**0001** 0010 0	0	**01**	0101	5	2
SWPB 6(R5)	1095 0006	**0001** 0000 1	0	**01**	0101	4	2

CALL 6(R5)	1295 0006	**0001** 0010 1	0	**01**	0101	5	2
SXT 6(R5)	1195 0006	**0001** 0001 1	0	**01**	0101	4	2
RRC EDE	1010 abs.add(EDE)–(PC+2)	**0001** 0000 0	0	**01**	0000	4	2
RRA EDE	1110 abs.add(EDE)–(PC+2)	**0001** 0001 0	0	**01**	0000	4	2
PUSH EDE	1210 abs.add(EDE)–(PC+2)	**0001** 0010 0	0	**01**	0000	5	2
SWPB EDE	1090 abs.add(EDE)–(PC+2)	**0001** 0000 1	0	**01**	0000	4	2
CALL EDE	1290 abs.add(EDE)–(PC+2)	**0001** 0010 1	0	**01**	0000	5	2
SXT EDE	1190 abs.add(EDE)–(PC+2)	**0001** 0001 1	0	**01**	0000	4	2
RRC &EDE	1012 abs.add(EDE)	**0001** 0000 0	0	**01**	0010	4	2
RRA &EDE	1112 abs.add(EDE)	**0001** 0001 0	0	**01**	0010	4	2
PUSH &EDE	1212 abs.add(EDE)	**0001** 0010 0	0	**01**	0010	5	2
SWPB &EDE	1092 abs.add(EDE)	**0001** 0000 1	0	**01**	0010	4	2
CALL &EDE	1292 abs.add(EDE)	**0001** 0010 1	0	**01**	0010	5	2
SXT &EDE	1192 abs.add(EDE)	**0001** 0001 1	0	**01**	0010	4	2
RETI	1300	**0001** 0011 0	0	**00**	0000	5	1

3. <u>פקודות Jump – ישנן 8 פקודות</u>: ערך של 4-bit MSB בין 0x2-0x3

חלוקת הקידוד לשדות

15 14 13	12 11 10	9 8 7 6 5 4 3 2 1 0
Op-code	C	10-Bit PC Offset

Instruction	Opc - Binary	C	Cycles	Length(W)
JEQ/JZ Label $\mathbf{If(flag\ Z\ ==\ 1)\quad PC_{new} = Label = PC_{old} + 2 + PC_{offset} \cdot 2}$	**001**	**0**01	2	1
JNE/JNZ Label $\mathbf{If(flag\ Z\ ==\ 0)\quad PC_{new} = Label = PC_{old} + 2 + PC_{offset} \cdot 2}$	**001**	**0**00	2	1
JC Label $\mathbf{If(flag\ C\ ==\ 1)\quad PC_{new} = Label = PC_{old} + 2 + PC_{offset} \cdot 2}$	**001**	**0**11	2	1
JNC Label $\mathbf{If(flag\ C\ ==\ 0)\quad PC_{new} = Label = PC_{old} + 2 + PC_{offset} \cdot 2}$	**001**	**0**10	2	1
JN Label $\mathbf{If(flag\ N\ ==\ 1)\quad PC_{new} = Label = PC_{old} + 2 + PC_{offset} \cdot 2}$	**001**	**1**00	2	1
JGE Label $\mathbf{If(flag\ N \oplus flag\ V\ ==\ 0)\quad PC_{new} = Label = PC_{old} + 2 + PC_{offset} \cdot 2}$	**001**	**1**01	2	1
JL Label $\mathbf{If(flag\ N \oplus flag\ V\ ==\ 1)\quad PC_{new} = Label = PC_{old} + 2 + PC_{offset} \cdot 2}$	**001**	**1**10	2	1
JMP Label $\mathbf{PC_{new} = Label = PC_{old} + 2 + PC_{offset} \cdot 2}$	**001**	**1**11	2	1

$$\mathbf{-512 \le PC_{offset} \le 511}$$

$$\mathbf{PC_{Next\ Instruction} = PC_{old} + 2}$$

<u>שימו לב:</u> מרחב ההסתעפות של פקודות jmp הוא בין PC-512words לבין PC+511words

כלומר בין PC-1024 לבין PC+1022

<u>Note</u>:

```
MOV    #LABEL,PC ; Branch to address LABEL
MOV    LABEL,PC  ; Branch to address contained in LABEL
MOV    @R14,PC   ; Branch indirect to address in R14
```