



שפת האסמבלי של מעבד ה-MIPS

מבוא ←

- מקבץ האוגרים (רגיסטרים)
- פקודות מסוג R
- פקודות מסוג I ופקודות מסוג J
- פסידו פקודות
- פעולות עם תווים
- אינטראקציה עם המשתמש באמצעות קלט ופלט
- לולאות, תנאים, פרוצדורות ומאקרואים

מבוא לשפת האסמבלי



כל מעבד מגיע עם ספר הנקרא Instruction Set המכיל בתוכו את כל הפקודות שהמעבד מבין. בהתאם לפקודות אלו הקומפיילר מתרגם את הפקודות בשפה עילית כלשהי לפקודות בשפת אסמבלי ואז האסמבלר של אותו מעבד מתרגם את הפקודות לפקודות בשפת מכונה שאותו מעבד מסוגל להבין.

בקורס שלנו, אנחנו לומדים את שפת האסמבלי ואת שפת המכונה של מעבד ה-MIPS בין השאר, מכיוון שסט הפקודות של מעבד זה יחסית מצומצם ופשוט ואפשר ללמוד אותו בסמסטר אקדמי. בשפת האסמבלי של מעבד ה-MIPS יש יחסית מעט פקודות, ואנחנו נצליח להסתדר עם הפקודות השכיחות והחשובות ביותר שבהן.

הפקודות במעבד ה-MIPS מתחלקות לשלושה סוגים שונים, בהמשך הפרק נלמד את סוגי הפקודות ונכיר את השוני ביניהם ואת המבנה של כל סוג פקודה.

בנוסף, חוץ מהפקודות הבסיסיות שהמעבד מבין, אסמבלרים שונים מציעים פקודות נוספות הנקראות "פסידו פקודות". מבחינת התחביר של הפקודה, הוא דומה מאוד לפקודה רגילה של המעבד, אך בפועל, הפקודה שבאמת רצה על המחשב שונה מזו שכתבנו באסמבלר. יש פסידו פקודות מסוימות שהופכות לשתי פקודות רגילות, ויש פסידו פקודות שפשוט הופכות לפקודה אחרת. כאשר מריצים תוכנית באסמבלר MARS, הוא מציג לנו את הפקודות שכתבנו, וליד כל פקודה כזו הוא מציג גם את הפקודה האמיתית שבאמת רצה. באסמבלרים שונים יכולים להיות פסידו פקודות שונות, וכן אסמבלרים שונים יכולים לתרגם פסידו פקודה אחת בצורה שונה זה מזה.

סט הפקודות המקורי של המעבד, בלי הפסידו פקודות, נקרא TAL - True Assembly.

ואילו סט הפקודות הכולל, גם עם הפקודות המקוריות וגם עם הפסידו פקודות, נקרא MAL - MIPS Assembly.

הפעולה הראשונה שהאסמבלר עושה בדרך ליצירת פקודות בשפת מכונה היא להפוך את כל הפקודות לפקודות השייכות לסט הפקודות המקורי של המעבד.

בשיעור זה נלמד כיצד לעבוד עם תוכנת MARS שהיא סימולטור של האסמבלר של מעבד ה-MIPS.

מבוא לשפת האסמבלי



ההסברים להלן נכונים לגבי מחשב עם מעבד MIPS:

זיכרון המחשב מורכב מהרבה מאוד תאים. בכל תא בזיכרון יש 8 ביטים (בית שלם). שני תאים יחד יכולים לאחסן חצי מילה (16 ביטים) וארבעה תאים יחד מאחסנים מילה (32 ביטים).

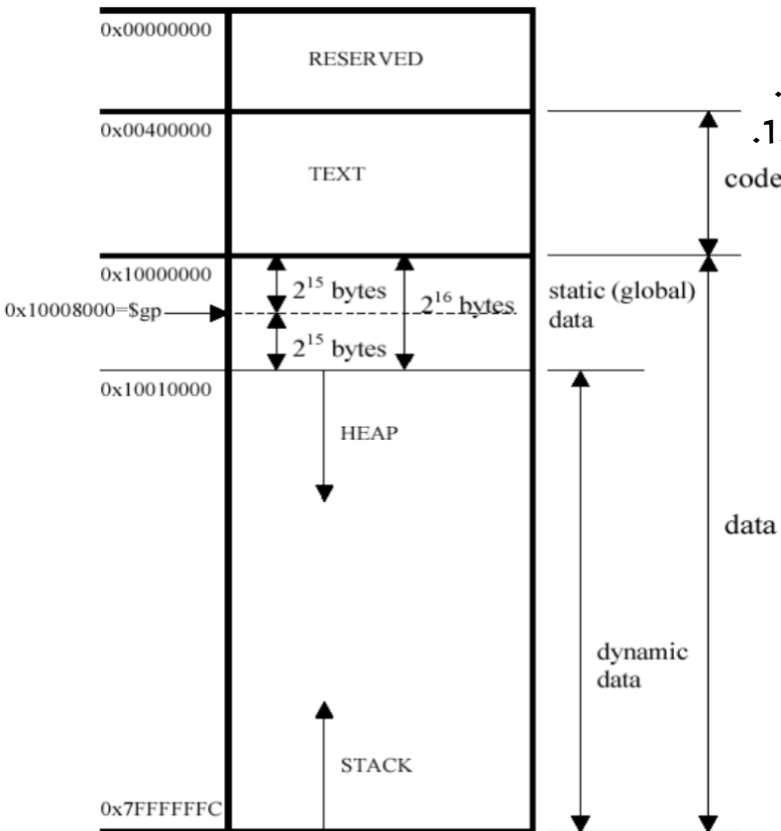
לכל תא בזיכרון יש כתובת המורכבת מ-32 ביטים.

לכן בזיכרון שלנו יכולים להיות 2^{32} תאים, שזה למעשה 4,294,967,296 בתים, או בקיצור 4Gbyte. אם נחלק את המספר שקיבלנו ב-4, נקבל את כמות המילים שאפשר לאחסן בזיכרון: 1,073,741,324.

כל פקודה בשפת מכונה במעבד ה-MIPS מורכבת מ-32 ביטים - מילה אחת. ולכן כל פקודה תופסת 4 תאים בזיכרון.

כאשר נרצה לכתוב או לקרוא מילה אל הזיכרון או מהזיכרון נצטרך לציין באיזו כתובת היא נמצאת בזיכרון. הכתובת חייבת להיות מספר המתחלק ל-4. בנוגע לקריאת וכתובת בתים או חצאי מילים, אפשר שהכתובת לא תתחלק ל-4, אבל אז נקבל נתונים מתא שלא נמצא בהכרח בתחילת המילה.

במחשבי MIPS הזיכרון מחולק לאזורים שונים, אזור שבו מאוחסנות הפקודות של התוכנית, אזור שבו מאוחסנים הנתונים ועוד. האזור של הנתונים מתחלק גם הוא לאזורים שונים, אזור של נתונים סטטיים ואזור של נתונים דינמיים. נלמד להשתמש באזור של המחסנית בהמשך.



מבוא לשפת האסמבלי



כאשר שומרים מילה או חצי מילה בזיכרון, יש שתי שיטות כיצד לאחסן אותה:

נגיד שאנחנו רוצים לשמור את המילה 0x43fe4c5a בכתובת 0x10000058

Big Endian		Little Endian	
כתובת בזיכרון	בית	כתובת בזיכרון	בית
0x10000058	0x43	0x10000058	0x5a
0x10000059	0xfe	0x10000059	0x4c
0x1000005a	0x4c	0x1000005a	0xfe
0x1000005b	0x5a	0x1000005b	0x43

למה זה חשוב? פשוט כי יש על זה לפעמים שאלות....

מבוא לשפת האסמבלי



קיימים ארבעה כללים שעומדים בבסיס התכנון של מעבד ה-MIPS ושל שפת האסמבלי שלו:

1. פשטות ואחידות -

- כל הפקודות באותו אורך (32 סיביות),
- השדות השונים בפקודות השונות דומים,
- קיימים מעט פורמטים לפקודות,
- כל הפקודות פשוטות וקצרות.

2. קטן זה מהיר -

- כל הפעולות האריתמטיות והלוגיות מתבצעות באמצעות רגיסטרים (וללא גישה לזיכרון),
- בפעולות אריתמטיות האופרנדים יכולים להיות קבוע (רק אחד) או רגיסטרים,
- קיימים 32 רגיסטרים בגודל של 32 סיביות כל אחד,
- מספר מוגבל של שיטות מיעון (addressing modes).

3. פשרה במקרים מסוימים -

- קיימים 3 סוגי פורמטים של פקודות וקיים שוני מועט במבנה שלהן.

4. הפיכת המצב השכיח למהיר -

- הרגיסטרים משמשים כאופרנדים של פעולות אריתמטיות,
- אפשר לבצע פעולה אריתמטית עם מספר קבוע קטן (עד 16 סיביות), ועבור מספרים גדולים צריך לבצע מספר פעולות.

מבוא לשפת האסמבלי



בשפת האסמבלי של מעבד ה-MIPS קיימים 3 סוגים של פקודות. לכל סוג כזה קיים מבנה פקודה טיפונת שונה מהאחרים. המבנה של הפקודה בשפת האסמבלי נראה בצורה מסוימת, והמבנה של הפקודה בשפת מכונה נראה בצורה אחרת

פקודות בשפת אסמבלי			
פקודות מסוג	מבנה סטנדרטי של פקודה	דוגמה	משמעות הדוגמה
R (register)	function \$rd, \$rs, \$rt	add \$t0, \$s1, \$s2	$t0 = s1 + s2$
I (immediate)	function \$rs, \$rt, immediate	beq \$t1, \$t2, label	אם הערכים ב-t1 וב-t2 שווים, קפוץ לכתובת של ה-label
J (jump)	function address	j label	קפיצה בלתי מותנית לכתובת של label

פקודות בשפת מכונה						
פקודות מסוג	6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
R (register)	opcode	rs	rt	rd	shamt	funct
I (immediate)		immediate (16 bit)				
J (jump)		address (26 bit)				

מילון					
שדה	פירוש		שדה	פירוש	
opcode	operation code	קוד פעולה	shamt	shift amount	מספר ביטים להזזה
rs	register source	אוגר מקור	funct	function	מספר פקודה בפורמט R
rt	register two	אוגר מקור שני	immediate		קבוע ב-16 ביט
rd	register destination	אוגר יעד	address		כתובת בזיכרון

מבוא לשפת האסמבלי



להתקנת האסמבלר MARS ניכנס לכאן:

<http://courses.missouristate.edu/kenvollmar/mars/download.htm>

נלחץ על הכפתור הירוק Download MARS.

במידה ולא מותקן על מחשבכם JAVA, יש להתקין אותו לפני: [/https://www.java.com/en/download](https://www.java.com/en/download)

במידה ויש לכם macOS Ventura 13 וקובץ ה-jar לא נפתח לכם, לחצו והחזיקו את הכפתור control ואז לחצו על הקובץ, דבר זה יפתח לכם תפריט, לחצו בתפריט על "פתח" ואז אשרו שאתם מעוניינים לפתוח את הקובץ. מעכשיו והלאה הקובץ ייפתח לכם תמיד בצורה רגילה.

אז בואו נכיר מעט את סביבת העבודה של ה-MARS...

מבוא לשפת האסמבלי



כמה כללים בסיסיים בנוגע לכתיבת קוד במרס:

1. נהוג להתחיל את התוכנית במספר הערות בנוגע לשם הקובץ, למהות שלו, לקלטים והפלטים שהוא מקבל ומוציא, ליוצר הקובץ ועוד... לדוגמה:

```
### File name: q3.s
### Developer: Steve Jobs
### Description: We receive two numbers from the user and calculate their product
```

2. במידת הצורך, אפשר להכריז על משתנים גלובליים לכל התוכנית בתחילתה, מיד לאחר ההנחיה: `.data`. המבנה של הכרזה על משתנה כדלהלן: `varName: .varType varValue`. לדוגמה:

```
.data
message1: .asciiz "\nPlease enter one number "
message2: .ascii "\nPlease enter one number "
var1: .byte 5
var2: .half 93
var3: .word 18594
var4: .float 23.4245
var5: .double 254.2452356
place1: .space 500

#מאחסן את המחרוזת במערך כתובות בזיכרון ומוסיף 0 בסיומה)
#(מאחסן את המחרוזת במערך כתובות בזיכרון)
#(מאחסן את הנתון(ים) ב-8 הביטים הפנויים הבאים)
#(מאחסן את הנתון(ים) ב-16 הביטים הפנויים הבאים)
#(מאחסן את הנתון(ים) ב-32 הביטים הפנויים הבאים)
#(מאחסן את הנתון(ים) כשבר עשרוני - נקודה צפה דיוק בודד)
#(מאחסן את הנתון(ים) כשבר עשרוני - נקודה צפה דיוק כפול)
#(שומר את כמות הבתים המצוינים הבאים בזיכרון)
```

3. לאחר ההכרזה על המשתנים מתחילים את הקוד עצמו, מיד לאחר ההנחיה: `.text`.

4. אפשר להוסיף תווית `main`: ולהגדיר אותה כגלובלית באמצעות ההנחיה: `globl main`. והאסמבלר יתחיל את התוכנית ממנה אם כך יוגדר לו `(Settings -> Initialize Program Counter to global 'main' if defined)`.

5. לאורך התוכנית אפשר ליצור תוויות (מילה בתחילת שורה המסתיימת בנקודתיים). תווית היא הכתובת בזיכרון של שורת הקוד המגיעה מיד אחריה. באמצעות תוויות אפשר לקפוץ בקלות לשורות קוד שונות בתוכנית.

6. שורה חדשה מעידה על פקודה חדשה (אין נקודה-פסיק בסוף שורה).

7. הערה נכתבת מיד אחרי סולמית.

8. תוכנית מסתיימת באמצעות הצבת הערך 10 באוגר `$v0` (אפשר באמצעות `li $v0, 10`) וקריאה לפקודה `syscall`