תקציר מספר 1

מבנה המחשב - ארכיטקטורה מנקודת ראות התוכניתן

כאשר מדברים על מבנה המחשב או ארכיטקטורה שלו, למעשה יש לנושא הזה שני הבטים, די שונים זה מזה.

1. מבנה המחשב כמכונה, מבחינה פיזית - כפי שמהנדסי החומרה שבנו אותו רואים אותו.
2. מבנה המחשב כמודל לתכנות - כפי שהתוכניתן רואה אותו.

בין 1 ל-2 חייבת להיות כמובן איזושהיא חפיפה - איכשהוא התוכניתנים צריכים להתחבר עם מה שמהנדסי חומרה יצרו. אך החפיפה הזו היא למעשה מאד מינימלית - ובכוונה כך. רצוי שהחלות של עבודת התוכניתנים בהחלטות של הבניה של המחשב כמכונה תהיה מינימלית - הדבר מאפשר פשטות של עבודת התכנות (הסבוכה בלאו הכי) וגם מאפשר שינויים של המחשב כגירסאות חדשות של ה-CPU. ואכן, בעוד שמבנה המחשב כמכונה השתנה מאד במעברים מה-8086 ל-286, 386, 486 ומשם לפנטיום, הרי שהמודל השתנה מעט מאד מאז ה-386.

בקורס הזה נתרכז כמעט בלעדית ב-2, מה שקרוי מודל התוכניתן. כפי שאפשר להבין ממה שמתואר לעיל, המודל אכן מיצג חומרה - אבל רק חלק קטן ממה שיש שם, וגם אז לא תמיד ברור איך המימוש הפיזי של אותם רכיבים נגישים. זה גם לא חשוב לתוכניתן וגם עשוי להשתנות במעבר מגירסה של מחשב לבא אחריו.

הרצת תוכנה במחשב

עם הפעלת המחשב מתבצעת תוכנית מינימלית הצרובה במחשב המחפשת בכתובות מסוימות מאד וידועות מראש על הדיסק הקשיח (או הדיסקט) איפוא נמצא הקובץ הביצועי - תוכנית (שיתואר להלן) שמאתחל את המחשב. המחשב מעביר את השליטה של המחשב לתוכנית הזאת על ידי העתקתה לזכרון וביצוע שלה - כאשר המשמעות של "ביצוע" יובהר בהמשך. מה שיקרה בהמשך הוא שתוכנית האתחול תבצע פעולות ותפעיל קבצים ביצועיים נוספים - לפי איך שתוכנתה.

התוכנית המינימלית, תוכנת האתחול, והקבצים הביצועיים שמופעלים אחר כך כתובים בשפת מכונה.

שפת מכונה

שפת מכונה - היא שפה פרטית של המחשב, שרק אותה יודע המחשב לבצע באופן ישיר.

תוכנית בשפת מכונה מתחלקת באופן גס לשטחי זכרון המשמשים לאכסון מידע מצד אחד ושטחי זכרון המתארים את חלק ביצועי מצד שני.

החלק הביצועי (או הפקודות) של התוכנית הוא למעשה נקודת המפתח של מימוש מחשבים ותוכנה כפי שאנחנו מכירים אותם. הבנת המרכיב הזה הוא הכרחי להבנה של כל מה שבא בהמשך.

כיצד בנוי החלק הביצועי?

החלק הביצועי הוא סידרה של פקודות מכונה.

לכל CPU קיימת סידרה של פקודות, שלהם יש מימוש מפורש בתוך ה-CPU. הפקודות הללו נקראות פקודות מכונה.

ב-8086 היו קצת מעל ל-מאה פקודות כאילו (לא כולל המעבד המתימטי). היום כאשר כוללים את המעבד המתימטי והגירסאות המתקדמות של מעבדי אינטל (386, 486, פנטיום) וה-MMX מגיעים היום לכמה מאות של פקודות מכונה. קשה לדעת כמה בדיוק - יש פקודות דומות או קרובות שקשה להחליט איך לספור אותם. גודל הפקודות הללו הם בין byte אחד למספר בתים. ב-8086/8 ששה בתים היה המקסימום ב-386 ואילך המקסימום הוא 13 בתים (לא כולל את האפשרות של בתים מקדימים prefix bytes שיכולים כל אחד להאריך את הפקודה בבית נוסף).

כל פקודת מכונה מבצעת למעשה פעולה מאד פשוטה באופן עקרוני - למשל חיבור, חיסור וכפל מספרים, העברת אינפורמציה ממקום אחד לשני וכו'. כל אחד מאתנו יכול לבצע כל פקודה כזו ביד. כאשר נראה את רשימת הפקודות הללו, ניוכח שאם מחשב עושה משהו חכם, כל החוכמה הזו נמצאת בתוכנה.

לכל פקודה כזו יש קוד מיספרי. גודל הקוד הזה באינטל x86 (עד כמה שידוע לי) הוא בין byte אחד לשלושה. כאשר ה-CPU נידרש לבצע תוכנית מסוימת הוא למעשה מעתיק לתוכו את אותה פקודת מכונה הנחשבת ל"ראשונה" של התוכנית הזו מהזכרון.

ה-CPU מזהה לפי הקוד איזה פקודת מכונה מדובר, ומבצע אותה על ידי המימוש המפורש של הפקודה המצוי בו. ביצוע הפקודה קובעת (בין השאר) מיהו (או איפוא נמצא) הפקודה הבאה לביצוע של התוכנית. הקוד של הפקודה הזו מועתקת לתוך ה-CPU והתהליך זיהוי - ביצוע - העתקה חוזר על עצמו עד שהמחשב נכבה או נעצר.

קובץ ביצועי EXE

קובץ ביצועי (Executable File) הוא קובץ בינארי הכתוב בשפת מכונה.

תחת DOS (ו-WINDOWS) הקבצים הללו באים אם סיומות EXE. (בדרך כלל) או COM. קובצי ה-COM. הם בעיקר ירושה של גירסאות הראשונות של DOS והשימוש בהם הולך ונעלם.

קובצי ה-BAT. הם משהו אחר לגמרי. אילו קובצי טקסט של פקודות DOS שמבוצעות באופן עקיף על ידי תוכנית אחרת.

לעומת זאת במערכת הפעלה Linux שגם היא רצה על מעבדי האינטל x86, לקובץ ביצועי לא חייב להיות סיומת מיוחדת - האבחנה בין קובץ ביצועי לקובץ אחר נעשה בדרך אחרת. במילים אחרות: העובדה שקובץ ביצועי ב-DOS חייב לבוא עם סיומת מסוימת היא תכונה של DOS ולאו דווקא של המחשב.

אוגרי ה-CPU

ה-CPU מכיל רכיבים הקוראים וכותבים מידע מהזכרון, המפענחים את הפקודה הנוכחית, מבצעים אותה וקובעים את הפקודה הבאה לביצוע. אבל ה-CPU גם מכיל רכיב נוסף שלכאורה לא היה הכרחי: יש לו זכרון מינימלי משלו, הנגיש לתוכנה, הנקרא אוגרי ה-CPU או בקיצור אוגרים (Registers). ב-8086 היו 14 אוגרים בני 16 ביט, ב-386 ואילך יש 34 אוגרים בני 32 עד 48 ביט. תאור מפורט של האוגרים הללו ינתן בהמשך. נציין כאן שבדרך כלל האוגרים הללו מתיחסים לפי שמות פרטיים שיש להם:
למשל האוגרים של ה-8086 נקראים AX, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES, IP, SP ו-FLAGS.

בכדי להבין מדוע צריך את האוגרים, צריך להכיר את העובדה הבאה: מה שקרוי הזכרון האלקטרוני (ה-RAM) של המחשב (שהיום הוא 32 - 16 מגהבייט ברוב ה-PC) יכול אך ורק לזכור: אפשר לכתוב לתוכו ולקרוא את הערך האחרון שנכתב לתוכו - וזה הכל. פעולות אריתמטיות נוסח חיבור או כפל חייבים איפוא להתבצע ב-CPU - ולשם כך קיימים האוגרים. חלק מהאוגרים נגישים לתוכנה - על מנת לאפשר מימוש קוד יעיל ככל האפשר.
השיקולים שמאחרי המבנה הזה נובעים מן הסתם משיקולים של להקל את מימוש הזכרון ורכיבי חומרה אחרים, ולא מן הנמנע הוא שהדבר עשוי להשתנות בעתיד. אבל עד כמה שידוע לי, המבנה הזה נכון לכל מחשב קיים.

שפת אסמבלי לעומת שפת מכונה

שפת מכונה היא שפה של מספרים בינאריים. ברור שקשה לתוכניתן אנושי לכתוב תוכניות בשפה כזו. לשם כך הומצא שפת האסמבלי שהוא מעין צורה סימלית - טקסטואלית של שפת מכונה:

באופן עקרוני, כל שורה בתוכנית אסמבלי מקביל לפקודת מכונה אחת וכל פקודת מכונה מקבילה לשורה אחת בשפת אסמבלי.

למשל פקודת המכונה בודדת המעבירה תוכן אוגר AX לתוך אוגר BX נכתוב בשפת אסמבלי:

```
MOV BX,AX
```

שפת אסמבלי היא מעין שפה סימלית מינימלית שמבטא את שפת המכונה. כמו ששפת מכונה היא שפה פרטית - יחודית לכל מחשב, גם שפת האסמבלי היא יחודית לכל מחשב. שפת האסמבלי של המעבד Digital Alpha הוא שונה ובלתי קומפטבילי לשפת האסמבלי של האינטל x86, למשל.

המעבד אינו יכול להריץ ישירות תוכנית בשפת אסמבלי, רק שפת מכונה. על מנת להריץ תוכנית בשפת אסמבלי צריך לבצע תהליך המרה דומה לזה של קומפילציה של תוכנית בשפת עילית. תהליך זה, שהוא מטבע הדברים פשוט יותר מקומפילציה, נקרא אסמבלי של התוכנית והתוכנית שעושה את התהליך הזה נקרא אסמבלר. ללמדך שהביטוי שהשתרש "שפת אסמבלר" הוא למעשה טעות.

מבנה של פקודות מכונה

שיקולים

כפי שתואר קודם, לכל פקודת מכונה שני תפקידים: לעשות משהו (כמו חיבור) ולקבוע את הפקודה הבאה לביצוע.

על מנת שפקודות המכונה יוכלו לבצע תוכניות כלליות, הם חייבים להיות בעלי יכולת לקרוא מגורם אחד במחשב (כתובת בזכרון למשל) לשנות גורם אחר. אחרת המחשב לא יוכל אפילו להעתיק מידע ממקום אחד למשנהו.

שיקולים נוספים (וסותרים במידה מסוימת) שמשפיעים על תכנון פקודות המכונה הם יעילות (שחשוב מאד כרמה הזו) והשלכות על מבנה ה-CPU. יעילות מתבטאת בכך שככל שמשימות נפוצות דורשות יותר פקודות מכונה לממש אותם, התוכניות (ביחוד אילו שכתובות בשפה עילית) ירוצו יותר לאט. לעומת זאת, ככל שנצפה

יותר מפקודת מכונה, הדבר יסבך את מימוש ה-CPU, דבר שעשוי ליקר אותו או לגרום לו לרוץ לאט יותר.

המבנה במקרה של ה-Intel x86

כפי שתואר לעיל, לכל פקודה יש קוד. הפקודה בנויה קודם כל ממנו. יש לו גם מרכיב נוסף, הנקרא אופרנד(ים).

אופרנד הוא תאור של גורם מידע שעליו פועלת הפקודה - למשל מספר שמקדמים אותו ב-1.
אופרנד יכול להיות ערך מיספרי (קבוע), התיחסות לאוגר או כתובת בזכרון.

במחשבי האינטל x86 יש, באופן עקרוני, פקודות של 2 אופרנדים, אופרנד אחד, או אפס אופרדים. יש פקודות שיש להם אופרנד או שני אופרנדים שמשתמעים מהפקודה כלומר שהפקודה תמיד פועלת על יעד מוגדר מראש. לדוגמא, כפל ב-8086 תמיד פועל על האוגר AX, ולכן אין צורך לציין זאת בפקודה.

פקודות 2 אופרנדים:

באופן כללי נוהגים בשפת אסמבלי לכתוב פקודת 2 אופרנדים בצורה:

שם הפקודה אופרנד יעד, אופרנד מקור

למשל

```
MOV AX,BX
```

מעבירה את תוכן BX (אופרנד מקור) לתוך AX (אופרנד היעד).
הפקודה:

```
ADD AX,BX
```

מסכמת את תוכן BX (אופרנד מקור) עם תוכן AX (אופרנד היעד) לתוך AX (אופרנד היעד).

הכללים הם כלהלן:

1. בפקודת 2 אופרנדים הפעולה יכולה להסתמך על ערכי שני האופרנדים, לפני ביצוע הפקודה, במידה וזה רצוי, כמו ב-ADD כדוגמא לעיל.
2. האופרנד שמשנה את ערכו הוא האופרנד היעד (השמאלי). באופן עקרוני, אופרנד המקור שומר על ערכו מלפני הפקודה. זה משהו כמו פקודת השמה בשפה עילית (x = y; , y שומר על ערכו, x משתנה).

אגב, לכלל הזה שרק האופרנד היעד משנה את ערכו יש לפחות חריג אחד - הפקודה ההחלפה XCHG המבצעת החלפה (Exchange) של שני האופרנדים:

```
XCHG AX,BX
```

יגרום ל-AX לקבל את תוכן BX ולהיפך: BX יקבל את תוכן AX, כלומר שני האופרנדים ישתנו. אבל האמור לעיל נכון ל(כמעט) כל פקודת 2 אופרנדים אחרת.

3. בפקודת 2 אופרנדים, אופרנד המקור יכול להיות אוגר, כתובת בזכרון או קבוע., מאופרנד היעד יכול להיות רק אוגר או כתובת בזכרון. הוא לא יכול להיות קבוע מספרי. ישנו רק מגבלה אחת: אין תמיכה למצב שבו שני האופרנדים הם כתובות בזכרון. לפיכך הצירופים האפשריים ל-< אופרנד מקור, אפרנד יעד > הינם:

< אוגר, אוגר>	למשל	MOV AX,BX
< זכרון, אוגר>	למשל	MOV AX,[BX]
< קבוע, אוגר>	למשל	MOV AX,9
< אוגר, זכרון>	למשל	MOV [BX],AX
< קבוע, זכרון>	למשל	MOV BYTE PTR [BX],9

<u>פקודת אופרנד אחד</u>:

באופן כללי מהצורה

אופרנד שם הפקודה

למשל, הפקודה

```
INC  AX
```

יקדם מספרית את תוכן AX באחד.
האופרנד היחיד עשוי לשנות את ערכו ועשוי שלא - בהתאם לאופי הפקודה.

<u>פקודה אפס אופרנדים</u>:

באופן כללי מהצורה:

שם הפקודה

למשל, הפקודה:

```
HLT
```

יגרום לעצירת המחשב (משתמשי Windows 95 גורמים לביצוע הפקודה הזו כאשר הם מבצעים shutdown של המחשב).

הזרימה של התוכנית - קביעת הפקודה הבאה לביצוע

כפי שנאמר קודם, הזרימה של תוכנית (כלומר איזה פקודות התוכנית מבצעת ובאיזה סדר) נקבעת על ידי כך שמבצעים את הפקודה "הראשונה" של התוכנית, ומרגע זה ואילך, כל פקודה קובעת עבור המחשב מיהו הפקודה הבאה לביצוע.
אפשר היה לתכנן את המחשב כדי שכל פקודת מכונה תקבל אופרנד נוסף: מיהו הפקודה הבאה לביצוע. אלא שברור שזה היה בלתי יעיל ובלתי אופטימלי מנקודת ראות של החומרה וגם מנקודת ראות של כתיבת התוכניות מהסיבה הפשוטה הבאה: בכל תוכנית, ברובם המכריע של הפקודות, הפקודה הרצויה להיות הבאה לביצוע היא הפקודה הבאה (העוקבת) בזכרון. רק בנקודות מסוימות מאד בתוכנית התוכניתן מעונין בהפרת הכלל הזה. לפיכך הוחלט ששינוי הפקודה הבאה לביצוע, לכתובת שאיננה הפקודה העוקבת בזכרון, יעשו על ידי פקודות מיוחדות שנועדו אך ורק לכך, הנקראות פקודות הסתעפות. דוגמאות לכך הו פקודת ה-JMP, פקודת ההסתעפות המותנית (JE, JNE) פקודות הסתעפות לפרוצדורה (CALL) פקודות הסתעפות לפסיקה (INT). למשל הפקודה

```
JMP label
```

תקבע את הפקודה הבאה לביצוע לפי הערך של האופרנד label (עוד נראה איך הדבר נעשה). הפקודה

```
JE label
```

תעשה אותו דבר אבל בתנאי מסוים (שעוד יוסבר). אם התנאי אינו מתקים, הפקודה הבאה לביצוע תהיה הפקודה העוקבת בזכרון.

במילים אחרות: עבור רוב פקודות המכונה, אלה שיש להם תפקיד שאינו קשור לזרימה של התוכנית כמו פעולות אריתמטיות (ADD, SUB, MUL) או העברת אינפומציה (MOV למשל) הפקודה הבאה לביצוע נקבעית באופן אוטומטי בתור הפקודה העוקבת בזכרון. במידה והתוכניתן מעונין שבנקודה מסוימת בתוכנית תהיה חריגה מהכלל הזה, הוא משתמש בפקודות הסתעפות.

סוגי פקודות מכונה

כפי שמשתמע מהאמור לעיל, פקודות המכונה מתחלקים לכמה סוגים גם לפי תפקידם (כלומר מה הם עושים). נהוג לסווג אותם כלהלן:

1. פקודות אריתמטיות (למשל ... ,ADD, SUB, MUL, DIV).

2. העברת אינפורמציה (... ,MOV, PUSH, POP, XCHG).

3. הסתעפות (JMP, JE, JG, CALL, INT, ...).

4. פעולות ביטיות (AND, OR, NOT, ...).

5. שליטה על המעבד (HLT, CLI, ...).

תוכנית דוגמא hellola.asm

תוכנית הדוגמא הראשונה שלנו תמחיש חלק משמעותי מהמבנה של תוכניות האסמבלי לפי השיטה שנעבוד בהם בשלב זה - תוכניות אסמבלי טהורות תוך שימוש בהנחיות הסגמנטים המקוצרות. בשיטה הזו האסמבלר פורש בשבילכם באופן אוטומטי הגדרות שונות ופותר אתכם מזה, וזה נוח בשלב המוקדם הזה.

- תזכורת: כל תוכנית הדוגמא כתובות בצורה כזו שכל מילה על טהרת האותיות הגדולות הם מילים שמורות בשפת האסמבלי. כל מילה שמכילה גם אותיות קטנות הם בחזקת מזהים שיכולים להיות מוחלפים בכל מילה אחרת.

- הנחיות לאסמבלר: מדובר בשורות בתוכנית שמגדירות לאסמבלר איך לפרש שורות פקודה שמתחתיהן אך אינם משתקפים ישירות בקובץ הבינארי.

- בכל שורה מה שנכתב לאחר ה-";" הינו <u>הערה</u>. האסמבלר פשוט יתעלם מכל תוכן שיש שם עד לשורה הבאה.

- ההנחיה

```
.MODEL SMALL
```

אומרת לאסמבלר לבחור ערכים מסוימים מבין אפשריות שונות שיש. לא ניכנס לכך בשלב זה.

- ההנחיה

```
.STACK 100h
```

מנחה את האסמבלר להגדיר מחסנית בגודל 100 הקסדצימל (256 עשרוני) בתים. למה זה נחוץ נלמד בהמשך הקורס. הגדרה שקולה לחלוטין תהיה:

```
.STACK 256
```

בכל הקשור לקבועים, אם התו האחרון בקבוע הוא H או D או B הקבוע

מפורש כהקסהדצימלי, עשרוני או בינארי בהתאמה. אם הוא על טהרת המספרים הוא יפורש כעשרוני. מספר הקסה שמתחיל באות (כמו 8800h) חייב להכתב עם אפס מוביל (0B800h בדוגמא) כי אחרת האסמבלר ינסה לפרש את המספר כשם של משתנה.

- ההנחיה

.DATA

אומר שמה שמתואר להלן יהיה תאור של שטח מידע - משתנים במושגים של שפת תכנות נוסח C. גם את התמונה המלאה לזה נראה מאוחר יותר.

- שורת הפקודה

```
DisplayString  DB 'Hello World!',13,10,'$'
```

הינה הגדרת <u>משתנה</u> בשם DisplayString שהוא למעשה מערך של תוים (בית בודד כל אחד).
הסיבה שמדובר במערך של בתים נקבעת על ידי ההנחיה DB. האפשרויות הם:

DB - בית אחד

DW - 2 בתים

DD - 4 בתים

DF, DP - 6 בתים

DQ - 8 בתים

DT - 10 בתים

באופן עקרוני איברים במערך מוגדרים בין פסיקים (...3,5,4,6) אבל במחרוזות ניתן להגדיר בין פסיקים ('Hello' שקול ל-'H','e','l','l','o'). האסמבלר מקצה מקום לקבועים הללו ודואג שיאותחלו במה שהמתכנת מצין.

הערכים 13,10 הם "עבור לשורה הבאה". ה-'$' הוא לצורכי סיום הדפסה - עוד נדבר על כך.

- ההנחיה

```
.CODE
```

מנחה את האסמבלי לכך שמה שמתואר להלן הוא החלק הביצועי של התוכנית.

- השורה

```
Begin:
```

הוא סמן (label) הנותן שם לפקודה. במקרה הזה אנחנו עושים זאת בכדי לסמן לאסמבלר מי הפקודה הראשונה לביצוע של התוכנית הזאת. הנקודה שבו מוכרות הפקודה הזו כראשונה לביצוע היא בהנחיה

```
END Begin
```

ה-END מנחה את האסמבלר שזהו סוף התוכנית ובמקרה הזה גם מציין ש-Begin הוא הפקודה הראשונה לביצוע של התוכנית.

- הפקודות

```
MOV AX,@DATA
MOV DS,AX
```

גורמים לאוגר המיוחד DS להצביע על ששח המשתנים של התוכנית (מה משמעות הדבר ולמה זה נחוץ נעמוד בהמשך).

- הפקודה INT 21h וקלט פלט

בתוכנית אסמבלי ביצוע קלט/פלט הוא לכאורה עינין טכני מאד הכרוך

בידעה מדויקת של מנגנון הקלט/פלט של המחשב אולם יש דרך להתחמק מכך וזה להסתמך על רוטינות קיימות במחשב. במקרה הזה אנחנו מסתמכים על מערכת ההבעלה DOS. INT 21h היא פניה לספרית רוטינות של DOS, הרוטינה המדויקת נקבעת לפי הערך של AH ברגע הקריאה. אנחנו מסתמכים על הרוטינות:

INT 21h, AH = 9

"הדפס למסך תוים מהנקודה DS:DX עד שתתקל בתו '$'".

INT 21h, AH = 4Ch

"סים ריצה והחזר שליטה ל-DOS".

הפקודה INT היא אחת מפקודות ההסתעפות (שמשנות את הפקודה הבאה לביצוע). היא פקודת הסתעפות די מיוחדת במספר מובנים, בין השאר שהיא מצינת איכשוא איך לחזור לתוכנית. INT 21h היא הסתעפות לשטח זכרון שמור ל-DOS שבו הוא מאכסן רוטינות קלט/פלט שלו. הרוטינות הללו משמשים את שורת הפקודה אך עומדות גם לרשות תוכנית אפליקציה.

בתוכנית אסמבלי עצירת התוכנית היא פקודה שהתוכנית חייבת לבצע אותה (שום דבר בתוכנית לא תבצע את זה אוטומטית). אם לא בצע את "פקודה החזרה" הזו, התוכנית תמשיך לקרוא זכרון ולנסות לפרש את התוכן כתאור של פקודות מכונה ולנסות לבצע אותם, דבר שבמקרה הטוב יתקע את התוכנית.

- הפקודה

MOV DX,OFFSET DisplayString

מציב ל-DX חלק מהכתובת של המשתנה DisplayString (ל-DS כבר דאגנו בשתי הפקודות הראשונות של התוכנית). זו איננה הדרך היחידה (או אפילו העיקרית) לחישוב כתובות, אנחנו נלמד על הפקודה LEA בשלב יותר מאוחר.

אם נניח ש- Var1 הוא שם של משתנה (2 בתים נניח) אז צריך להבדיל

בין

```
MOV AX,OFFSET Var1
```

שמציבה ל-AX את הכתובת של Var1 לבין

```
MOV AX,Var1
```

המציבה ל-AX את התוכן של Var1. זה בערך כמו ההבדל בין x = &y; לבין x = y; בשפת C.

- הידור התוכנית

בשלב הזה, אחרי שנקליד תוכנית נוסח hellola.asm באמצעות תוכנת עריכה (editor) נתרגם אותם לקובץ exe בשני שלבים, תוך יצירת קובץ ביניים עם סיומת obj:

```
tasm hellola.asm
```

ו-

```
tlink hellola.obj
```

ההרצה עצמה תהיה הרצת הקובץ hellola.exe.

במידה ויש שגיאות ה-tasm יודיע על כך ולא יווצר קובץ ה-obj.

אם אנחנו רוצים להשתמש ב-Turbo Debugger בכדי לאתר שגיאות נריץ:

```
tasm /zi hellola.asm
tlink /v hellola.obj
td hellola.exe
```

```
;
;  hello1a.asm - send message 'Hello World!' to the screen.
;
     .MODEL SMALL
     .STACK 100h
     .DATA
DisplayString DB 'Hello World!',13,10,'$'
                              ;
     .CODE
Begin:
     MOV AX,@DATA      ; DS can be written to only through a register
     MOV DS,AX         ; Set DS to point to data segment
     MOV AH,9          ; Set print option for int 21h
     MOV DX,OFFSET DisplayString  ;  Set  DS:DX to point to DisplayString
     INT 21h                ;  Print DisplayString
;
     MOV AH,4Ch        ; Set terminate option for int 21h
     INT 21h        ; Return to DOS (terminate program)
     END Begin
```

```
E:\>tasm hello1a.asm
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland International

Assembling file:   hello1a.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  389k


E:\>tlink hello1a.obj
Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

E:\>hello1a.exe
Hello World!

E:\>
```

תוכנית דוגמא hello4.asm

זוהי תוכנית דומה עקרונית ל-hellola.asm אך שואלת את המשתמש אם השעה היא אחרי 12 בצהריים ולפי תגובת מהמשתמש מדפיסה או את "Good afternoon world!" או "Good morning world!". אם התגובת המשתמש היא 'y' או 'Y' יודפס "Good afternoon world!" ובמקרה של כל תגובה אחרת יודפס "Good morning world!". על מנת שהתוכנית לא תבדיל בין 'y' ל-'Y' היא חייבת לכצע את שתי ההשוואות.

בנוסף לרוטינות INT 21h עליהם הסתמכנו ב-hellola.asm אנחנו מסתמכים על הרוטינה:

INT 21h, AH = 1

שמשמעותה "המתן ללחיצת מקש (עם קוד Ascii) מהמקלדת והחזר את קוד ה-Ascii ב-AL".

סימו לב למבנה של קטעי התוכנית

```
   MOV AH,1
   INT 21h
   CMP AL,'y'
   JE IsAfternoon

     .....

   MOV DX,OFFSET GoodMorningMessage
   JMP DisplyGreeting
IsAfternoon:
   MOV DX,OFFSET GoodAfternoonMessage
DisplayGreeting:
   MOV AH,9
   INT 21h
```

כאן אנחנו למעשה רואים פתות או יותר איך בפועל ממומשים תוכנית

בשפת מכונה ובאסמבלי: כל הפקודות שעשויות להתבצע נמצאות בגוף התוכנית. פקודות ההסתעפות של התוכנית דואגות לכך שה-CPU יעקוף את הפקודות שלפי הנסיבות אמורות שלא להתבצע ולהגיע אל אלה שכן. הדבר נעשה ע"י שילוב של פקודת השוואה (CMP) ופקודות הסתעפות (JE, JMP). כמושגים של שפת C זה כאילו יש בשפה רק "if" ו-"goto" אבל אין מבנים כמו "{" ו-"}" שלא לדבר על while וכו'. אפקט ה-"if" מתקבל ע"י פקודות הסתעפות מותנות שבו שינוי הפקודה הבאה מתבצעת רק אם תנאי מסוים מתקיים, ואחרת הפקודה הבאה לביצוע היא הפקודה העוקבת בזכרון. בתוכנית הזו משתמשים בפקודה "JE" שמשמעותה "הסתעף במקרה של שיוון". המבנה שמדובר כאן הוא שהפקודה

```
'CMP AL,'y
```

משווה את תוכן AL עם 'y' ושומר את התוצאה היכן שהוא. הפקודה

```
JE IsAfternoon
```

מבצעת את הסתעפות לנקודה האמורה (שינוי הפקודה הבאה לביצוע) רק אם הם אכן שוים. במידה ולא הפקודה הבאה לביצוע תישאר הפקודה העוקבת בזכרון, ואז מתבצעים הפקודות העוקבות תוך עקיפת הפקודות של המקרה השני על ידי פקודת ההסתעפות הבלתי מותנית JMP.

```
;
; hello4.asm - Conditional response.
;
     .MODEL SMALL
     .STACK 100h
     .DATA
TimePrompt DB 'Is it after 12 noon (y/n)?',13,10,':$'
GoodAfternoonMessage DB 13,10
         DB 'Good afternoon, world!',13,10,'$'
GoodMorningMessage DB 13,10
         DB 'Good morning, world!',13,10,'$'
                              ;
                              ;
     .CODE
ProgStart:
     MOV AX,@DATA                ; DS can be written to only through a register
     MOV DS,AX                   ; Set DS to point to data segment
     MOV AH,9                    ; Set print option for int 21h
     MOV DX,OFFSET TimePrompt    ;  Set  DS:DX to point to TimePrompt
     INT 21h                     ;  Print TimePrompt
     MOV AH,1                    ; DOS get character function #
     INT 21h                     ; Get a single character from keyboard
     CMP AL,'y'                  ; AL has input. Compare with 'y'
     JE  IsAfternoon             ; If AL = 'y' then go to IsAfternoon
     CMP AL,'Y'                  ; Compare with 'Y'
     JE  IsAfternoon             ; If AL = 'Y' then go to IsAfternoon
IsMorning:
     MOV  DX,OFFSET GoodMorningMessage  ; Point display message to morning
     JMP  DisplayGreeting               ; Avoid following code
IsAfternoon:
     MOV  DX,OFFSET GoodAfternoonMessage ; Point display message to afternoon
DisplayGreeting:
     MOV AH,9                           ; Set print option for int 21h
     INT 21h                            ; Print  chosen message
     MOV AH,4Ch                  ; Set terminate option for int 21h
     INT 21h                     ; Return to DOS (terminate program)
    END ProgStart
```

```
E:\>tasm hello4.asm
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland International

Assembling file:    hello4.asm
Error messages:     None
Warning messages:   None
Passes:             1
Remaining memory:   376k


E:\>tlink hello4.obj
Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

E:\>hello4.exe

Is it after 12 noon (y/n)?
:y
Good afternoon, world!

E:\>
```