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 שהתלות 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 בגירסאות 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 - 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 הבעה 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 ברמה 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:

אופרנד מקור, אופרנד יעד שם הפקודה

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 בדוגמא 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 <u>שני</u> 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, ...).