בס"ד

**עקרונות שפות תוכנה – תרגול 1**

**Principles of Programming Languages**

**אתר הקורס עליו מבוסס הקורס הזה נקרא From Nand to Tetris, וניתן למוצאו בלינק** <https://www.nand2tetris.org/>.

**באתר ניתן למצוא:**

* מצגות מפורטות לגבי כל תרגיל (בטאב שנקרא Projects)
* פירוט דרישות עבור כל חמשת הפרויקטים שיוגשו במהלך התרגול (בטאב שנקרא Projects)
* סריקה של חלק מפרקי הספר שעליו מבוסס האתר (בטאב שנקרא Projects)
* הורדת כל החומרים והתוכנות הרלוונטיים לביצוע התרגילים (בטאב שנקרא Software)

**ספר הקורס עליו מבוסס האתר נקרא:**

* **The Elements of Computing Systems: Building a Modern Computer from First Principles**, By [Noam Nisan](http://www.cs.huji.ac.il/~noam/) and [Shimon Schocken](http://www.faculty.idc.ac.il/schocken/)
* הספר ניתן להשאלה בספריה.

**האתר והספר בנויים מ 12 תרגילים/פרויקטים. אנחנו בקורס שלנו נלמד רק את פרק 4 ופרקים 7-11. התרגילים יתבססו רק על פרקים אלה.**

**התרגילים (בניית קומפיילר):**

במהלך התרגול, כל זוג בעזרת השפה הייחודית שקיבל, יבנה **קומפיילר** לשפה עילית בשם **Jack**.

הקומפיילר **יקבל כקלט מחרוזת שהיא מסלול של ספריה** המכילה קבצי טקסט \*.jack)) המכילים קוד בשפה העילית Jack.

לאחר מכן, הקומפיילר יתרגם כל קובץ jack לקובץ בשפת הביניים VM (\*.vm).

לבסוף, הקומפיילר יתרגם את כל קבצי ה-VM לקובץ אחד בשפת assembly בעל סיומת asm.

את הקומפיילר נבנה בעזרת חמישה תרגילים.

**One or more** file(s)

(\*.vm)

Written in VM language

**Only One** file

(\*.asm)

Written in Hack assembly language

**One or more** file(s)

(\*.jack)

Written in JACK language

תרגיל 3 (פרק 9 בספר) – תרגיל קצר להכרת שפת JACK

תרגיל 4 (פרק 10 בספר)

תרגיל 5 (פרק 11 בספר)

תרגיל 1 (פרק 4 ופרק 7 בספר)

תרגיל 2 (פרק 8 בספר)

תרגיל 0 – תרגיל חימום קצר להכרת שפת התכנות האישית של כל זוג

**תרגיל 1 (פרוייקט 07)**

לצורך תרגיל 1 נצטרך להכיר שתי שפות וסביבות עבודה חדשות: שפת ביניים בשם **VM**ושפת **assembly** למכונה **HACK.**

* מצגות PDF המפרטות את חומר הרקע לתרגיל 1 ניתן לראות באתר הקורס:

<https://drive.google.com/file/d/19fe1PeGnggDHymu4LlVY08KmDdhMVRpm/view>

* הצעה לעיצוב התוכנה של תרגיל 1 ניתן למצוא בסעיף 7.3.2 בפרק 7 בספר.
* הגדרה מדוייקת של דרישות תרגיל זה, כולל כל הקבצים שיש להוריד לצורך התרגיל, ניתן למצוא באתר הקורס: <http://nand2tetris.org/07.php>

**האמולטורים, סביבות העבודה להרצת התרגילים (התרגומים):**

* **VMEmulator**, אמולטור המדמה את סביבת ה VM
* **CPUEmulator**, אמולטור המדמה את סביבת ה ASM HACK

**הורדה של כל החומר הנדרש לכל התרגילים וכן של האמולוטורים והמדריכים ללימוד האמולטורים:**

* יש להיכנס לאתר הקורס לטאב שנקרא Software

<http://nand2tetris.org/software.php>

* שם יש ללחוץ על הקישור הבא:

[Download the Nand2tetris Software Suite](https://drive.google.com/open?id=1xZzcMIUETv3u3sdpM_oTJSTetpVee3KZ) Version 2.6 (about 730K).

ולהוריד בעזרתו קובץ ZIP שנקרא nand2tetris.zip שמכיל את כל החומר הנדרש לתרגול בקורס זה. נסביר על תוכן הקובץ בשיעור התרגול.

* לאחר הורדת קובץ הZIP:
* **Mac users:** after downloading, read this [Setup Guide for Apple MacOS](https://drive.google.com/open?id=1QDYIvriWBS_ARntfmZ5E856OEPpE4j1F), written by Yong Bakos.
* **Windows users:** after downloading, put the downloaded zip file in an empty directory on your computer, and extract its contents as is, without changing the directories structure and names.
* In order to use the nand2tetris software tools, your computer must be equipped with a Java Run-time Environment. The JRE can be downloaded freely from many sites including [this one](http://java.com/en/download/index.jsp). For best performance, download the latest available version.

**לאחר ההורדה וביצוע ה extract, ניתן להריץ את האמולטורים בלחיצה על הקבצים הבאים שימצאו תחת הספריה tools שנמצאה ב ZIP:**

* VMEmulator.bat
* CPUEmulator.bat

**במחשב עם Mac או מחשב עם מערכת הפעלה Linux, ניתן להריץ את האמולטורים ב-bash scripts הבאים:**

* VMEmulator.sh
* CPUEmulator.sh

**האמולטורים עושים שימוש בטיפוסי קבצים הבאים:**

* **\*.tst** – scripts to run using **CPUEmulator**
* **\*VME.tst** – scripts to run using **VMEmulator**
* **\*.cmp** – comparison files, used by the **VMEmulator** and **CPUEmulator**
* **\*.out** – output file of the **VMEmulator** and **CPUEmulator**

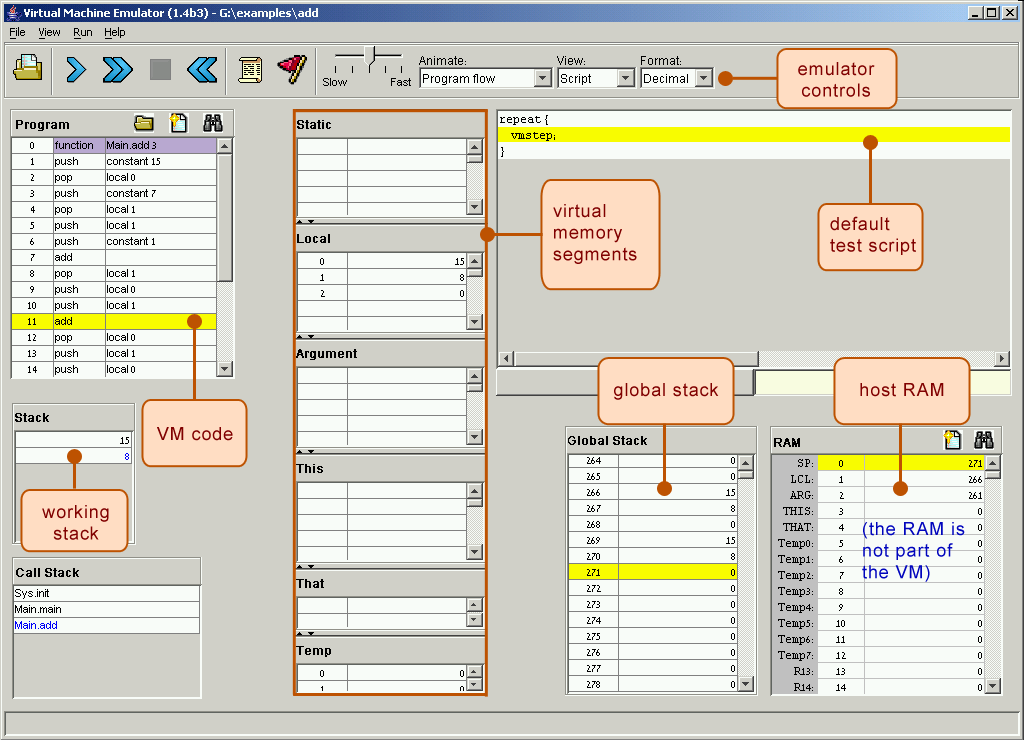
**The VM language** and the **Virtual Machine (פרק 7 בספר)**

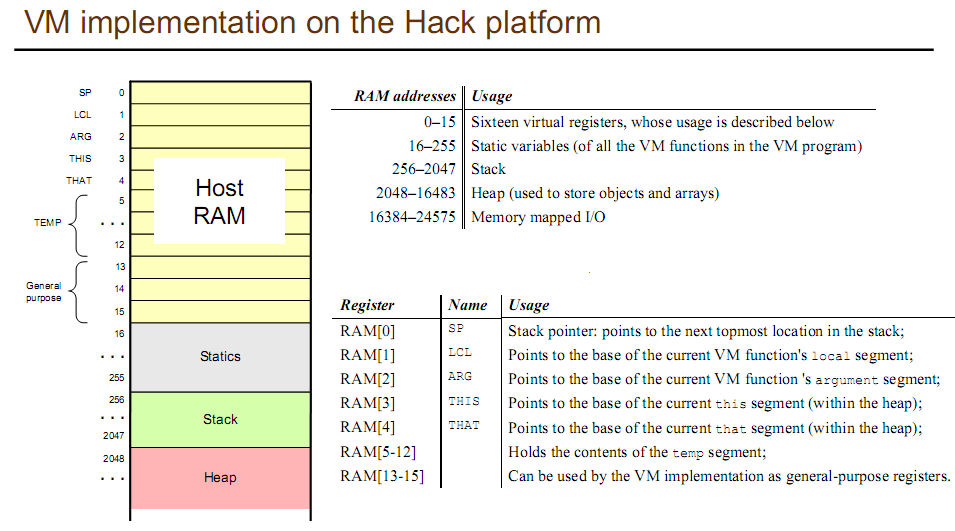
שפת VM היא שפה המבוססת על **מחסנית**. כל הפעולות נעשות על המחסנית בלבד.

**נדגים פעולות על המחסנית:**



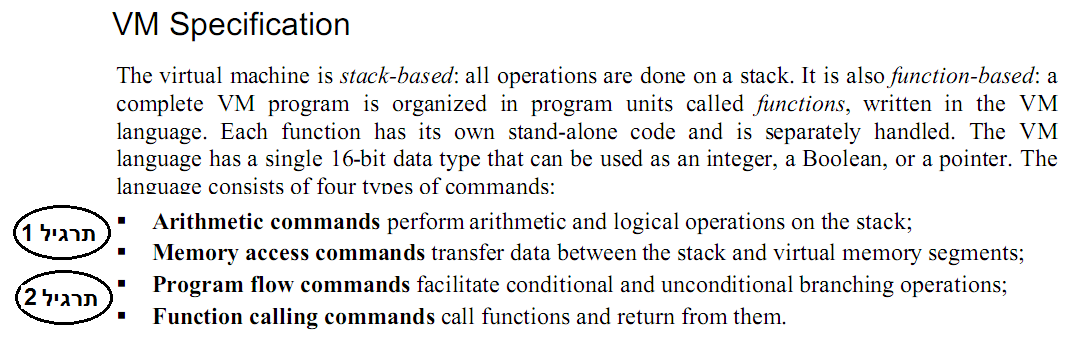
**נריץ את ה VMEmulatorבלחיצה על הקובץ VMEmulator.bat ונכיר את סביבת VM:**

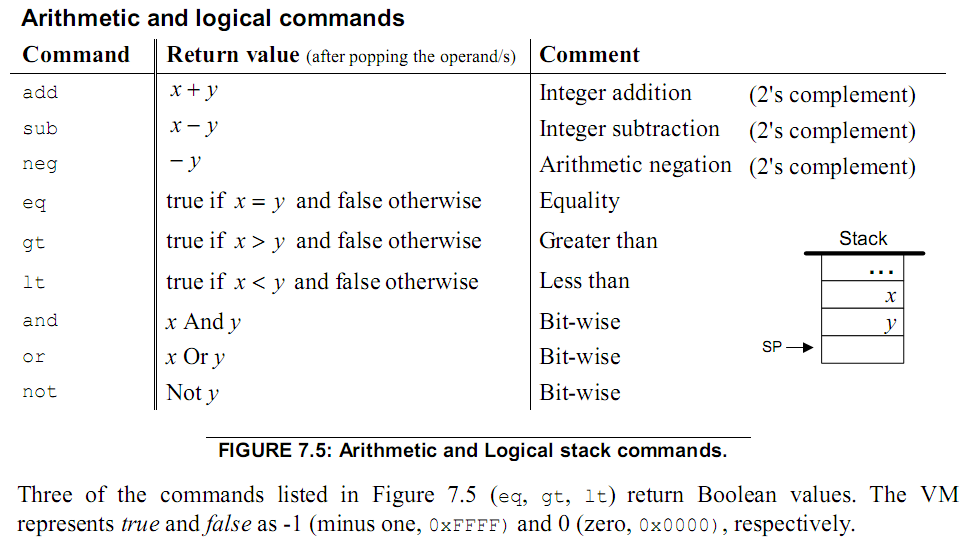


****

**בכיתה נראה כיצד מריצים קובץ עם פקודות VM, תוך כדי הסברים על האמולטור.**

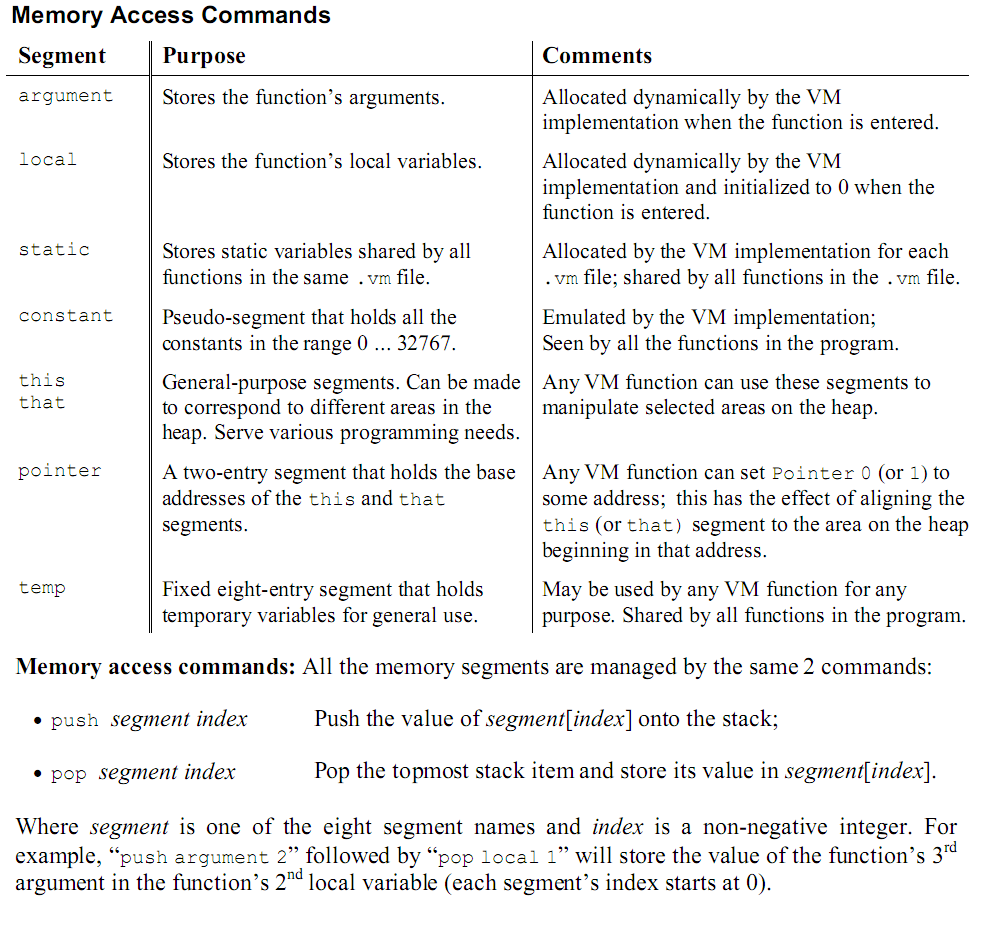
**פירוט הפקודות בשפת VM:**

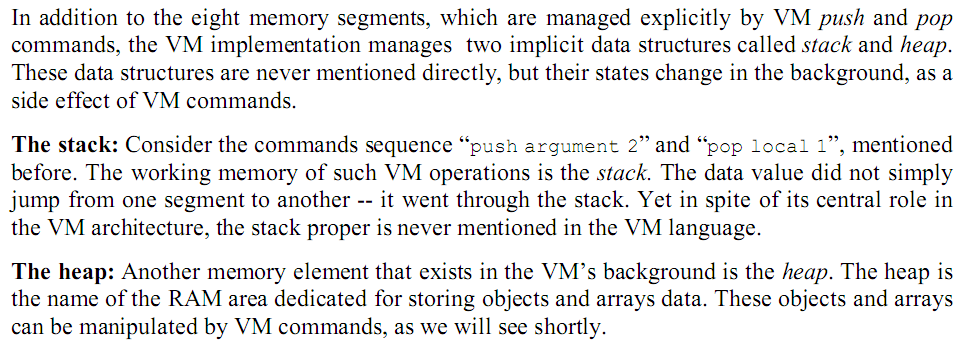


****

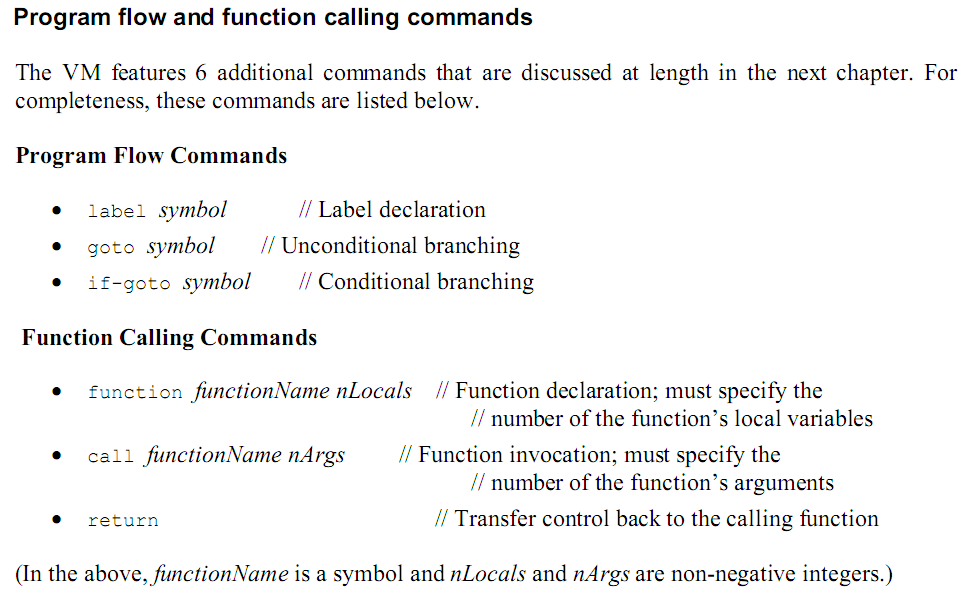
Since VM is **stack-based**, for **arithmetic and logic** commands:

* The binary commands mentioned above, work on the 2 operands as followed: **pop** the 2 top-most elements from the stack and **push** their arithmetic/logic result to the stack.
* The **'not'** and **'neg'** commands, are unary operations and work as followed: **pop** the top-most element from the stack and **push** its negation to the stack.

****

****

**את קבוצות הפקודות הבאות של שפת VM נתרגם בתרגיל 2:**

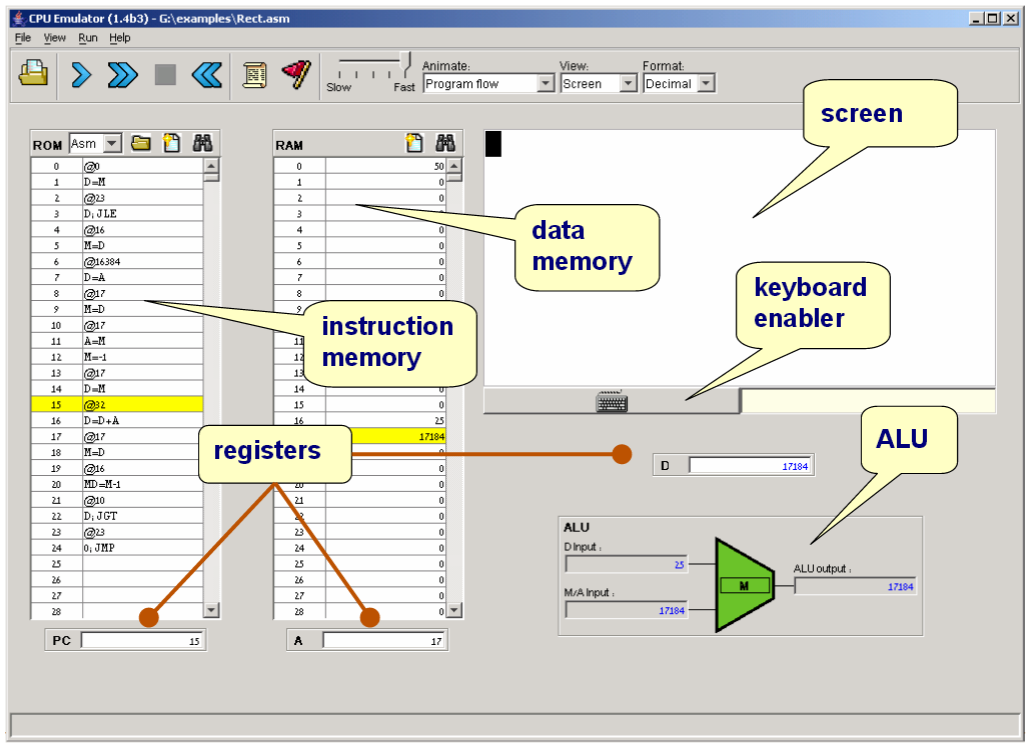


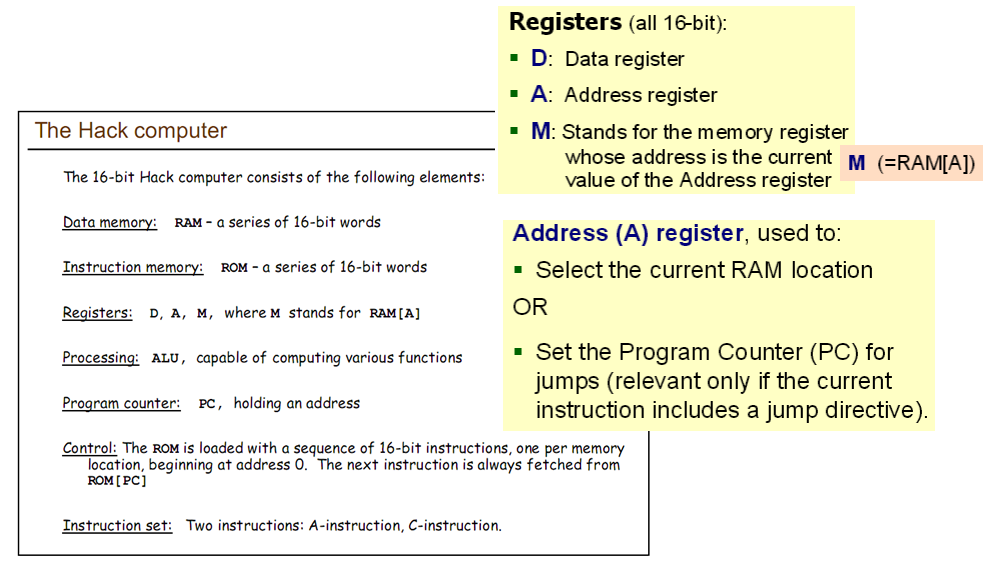
**פירוט של פקודות POP PUSH:**

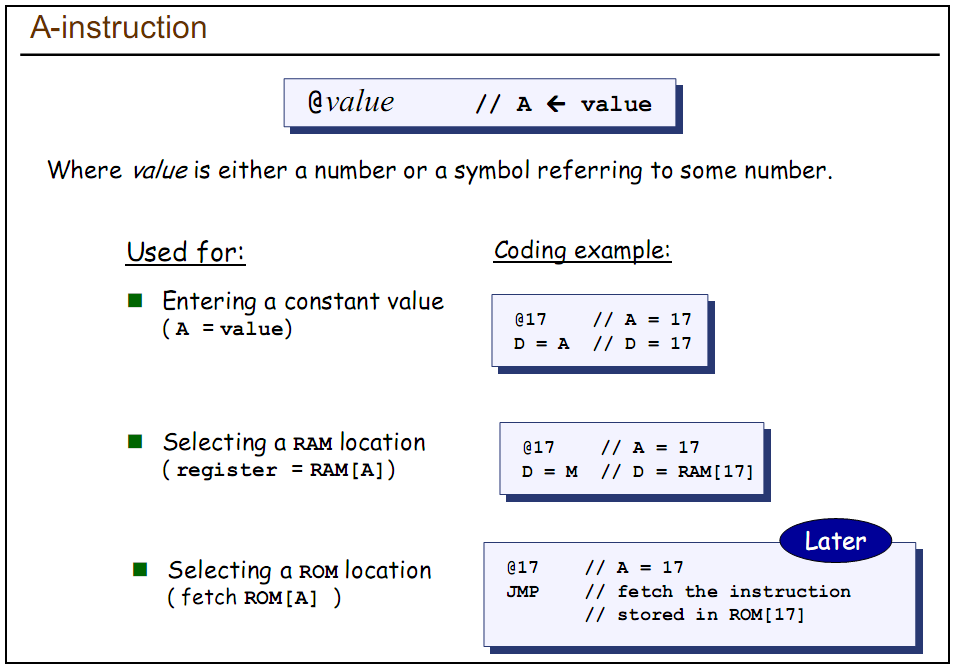
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Command** | | | **Meaning** | | | | **Remarks** |
| **Group 1** (local, argument, this, that) | | | | | | | |
| pop **local x** | | | **pop** the top of the stack into address RAM[ RAM[**LCL**] + **x** ] | | | | LCL =1 |
| push **local x** | | | **push** into the top of the stack the value that is in address RAM[ RAM[**LCL**] + **x** ] | | | |
| pop **argument x** | | | **pop** the top of the stack into address RAM[ RAM[**ARG**] + **x** ] | | | | ARG =2 |
| push **argument x** | | | **push** into the top of the stack the value that is in address RAM[ RAM[**ARG**] + **x** ] | | | |
| pop **this x** | | | **pop** the top of the stack into address RAM[ RAM[**THIS**] + **x** ] | | | | THIS =3 |
| push **this x** | | | **push** into the top of the stack the value that is in address RAM[ RAM[**THIS**] + **x** ] | | | |
| pop **that x** | | | **pop** the top of the stack into address RAM[ RAM[**THAT**] + **x**] | | | | THAT =4 |
| push **that x** | | | **push** into the top of the stack the value that is in address RAM[ RAM[**THAT**] + **x** ] | | | |
| **Group 2** (temp) | | | | | | | |
| pop **temp x** | **pop** the top of the stack into address RAM[ **5** + **x** ] | | | | **5** is constant value, since **temp** variables are saved on RAM[5-12] | | |
| push **temp x** | **push** into the top of the stack the value that is in address RAM[ **5** + **x** ] | | | |
| **Group 3** (static) | | | | | | | |
| pop **static x** | | **pop** the top of the stack into address RAM[**className.x** ] | | **All static** variables are saved on RAM[16-255]  Each class has its own static variables.  Each class should have separate allocation memory for its own static variables inside RAM [16-255]  **See more details at the last page of this document.** | | | |
| push **static x** | | **push** into the top of the stack the value that is in address RAM[**className.x** ] | |
| **Group 4** (pointer 0, pointer 1) | | | | | | | |
| pop **pointer 0** | | **pop** the top of the stack into address RAM[**THIS**] | | | | | Pointer 0 is THIS.  THIS =3 |
| push **pointer 0** | | **push** into the top of the stack the value that is in address RAM[**THIS**] | | | | |
| pop **pointer 1** | | **pop** the top of the stack into address RAM[**THAT**] | | | | | Pointer 1 is THAT.  THAT =4 |
| push **pointer 1** | | **push** into the top of the stack the value that is in address RAM[**THAT**] | | | | |
| **Group 5** (constant) | | | | | | | |
| push **constant x** | | | **push** into the top of the stack the value **x** | | | There is no 'pop' command in constant segment. | |

**The Hack computer & the Hack assembly language (פרק 4 בספר)**

**נריץ את ה CPUEmulatorבלחיצה על הקובץ CPUEmulator.bat ונכיר את סביבת Hack:**





****

**@value:**

The value is either a **positive number** (as shown above) or a **symbol** referring to some number.

Let's see examples for using a **symbol** as @value:

//sum = 0

@sum // sum refers to some memory location.

// for example memory location 20

M = 0 //M = RAM[A] = 0 🡺 RAM[20] = 0

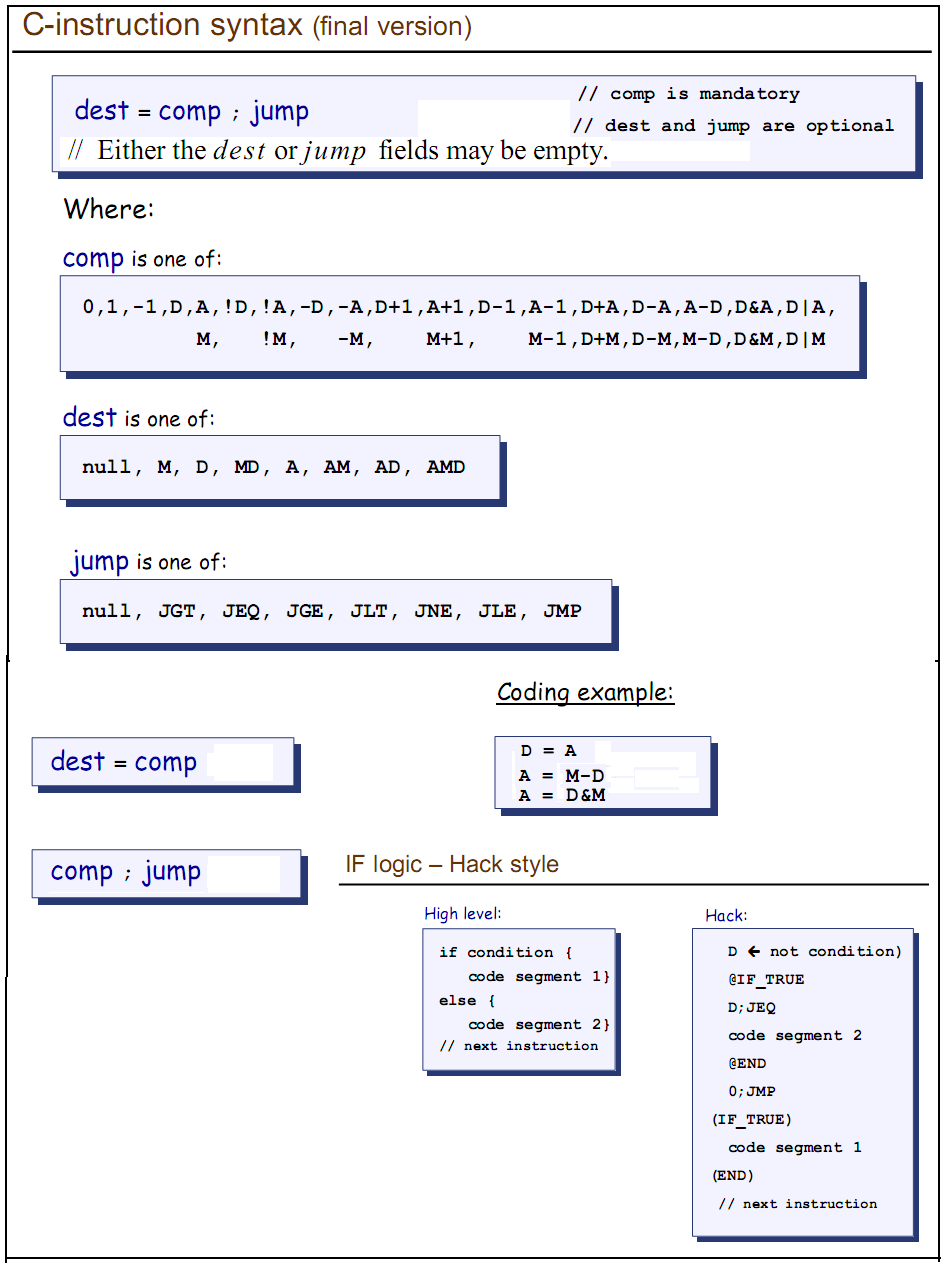
// i++

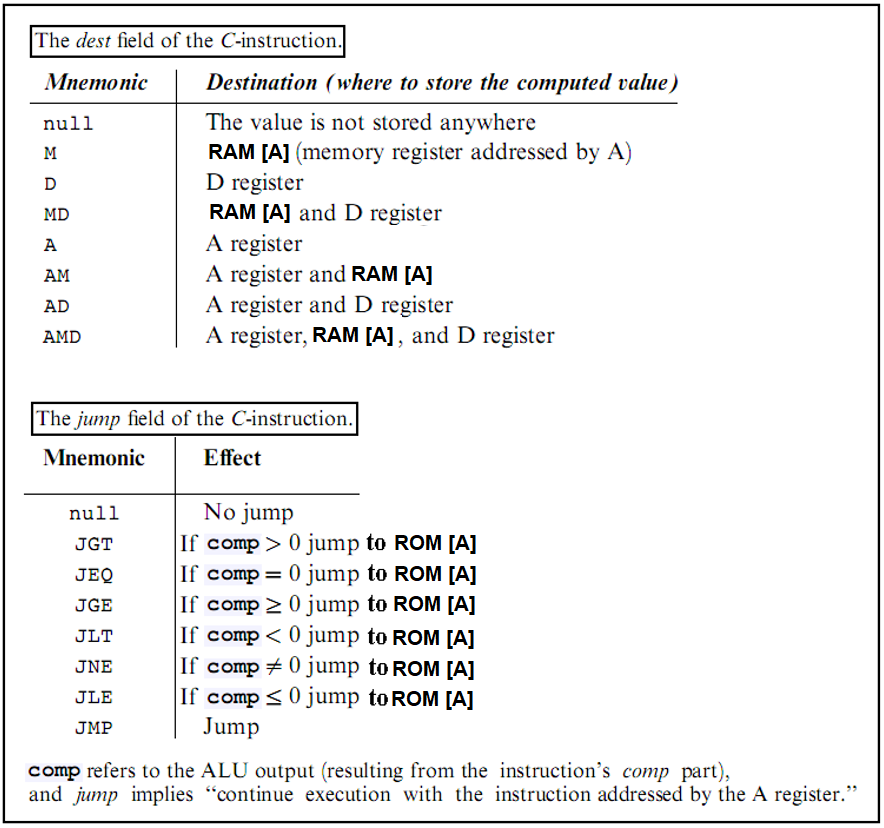
@i // i refers to some memory location.

// for example memory location 25 with value 3

D = M // D=RAM[A] = RAM[25] = 3

M = D+1 //RAM[A] = 3+1 = 4





|  |  |
| --- | --- |
| **C-instruction summary** | |
| dest = comp; jump | 1. Compute the **comp** field 2. Put its results into **dest**. 3. Compare its result to **0 upon the jump type** 4. If true **jump to ROM[A]**, else continue to the next instruction. |
| dest = comp | 1. Compute the **comp** field 2. Put its results into **dest**. |
| comp; jump | 1. Compute the **comp** field 2. Compare its result to **0 upon the jump type** 3. If true **jump to ROM[A]**, else continue to the next instruction. |

**Static variables:**

לפני שנתחיל להסביר על ה static segment, נסביר על **סימבולים** בשפת ASM:

**סוג ראשון: הקבועים SP, LCL, ARG, THIS, THAT**

SP is the constant 0

@SP // is like @0

LCL is the constant 1

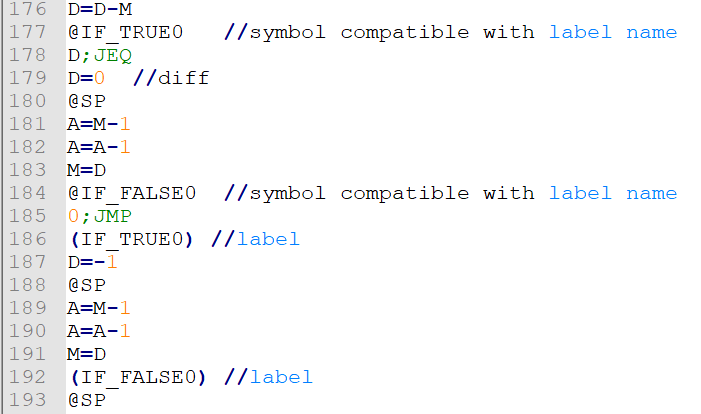
ARG is the constant 2

THIS is the constant 3

THAT is the constant 4

**סוג שני: תוויות (labels) שהמתכנת הוסיף ומשתמש בשילוב עם פקודות JUMP**

למשל בפקודות אריתמטיות עם תנאי. הסימבול של הלייבל יתורגם למספר הפקודה ב ROM שנמצאת מיד לאחר הלייבל. לפני הקפיצה יש לטעון את הסימבול הזה לרגיסטר A ואז הקפיצה תתבצע למספר הפקודה הזו.

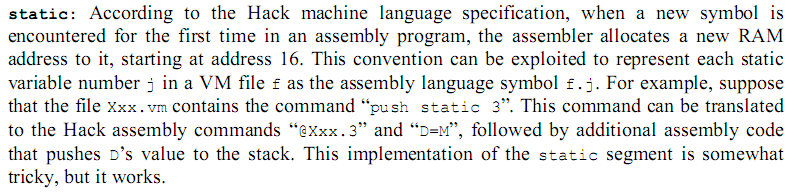


**סוג שלישי: סימבולים שהמתכנת הוסיף כדי להתייחס לשדה סטטי של מחלקה מסוימת.**

ועל כך נסביר כעת (למטה).

כאשר האמולטור טוען קובץ ASM והוא פוגש סימבול שאינו מהסוג הראשון או השני אז הוא ממספר אותו באופן אוטומטי כ16. (16 זה תחילת האזור הסטטי ב HACK). כשהוא פוגש סימבול נוסף שאינו מהסוג הראשון והשלישי הוא ממספר אותו כ 17 וכן הלאה.

עלינו לדאוג בתרגום להרכיב סימבול ייחודי לכל שדה סטטי של כל מחלקה. כך שלא ידרסו זה את זה.

****

In VM:

Each **vm file** describes one **vm class**.

Each **class** has its own **static** variables.

In Hack:

**All static** variables are saved on RAM[16-255]

Each class should have **separate allocation memory** for its own static variables inside RAM [16-255]. **How?**

By giving a full name for each static variable, which compounds from the vm class (file) where it is defined and its index inside this class (file).

**For example:**

|  |  |
| --- | --- |
| The content of first file 'ClassA.vm' is: | The content of second file 'ClassB.vm' is: |
| //…  push static 0  //…  push static 1  //…  pop static 0  //…  pop static 1 | //…  push static 0  //…  push static 1  //…  pop static 0  //…  pop static 1 |
| The translation from **'ClassA.vm'** to asm | The translation from **'ClassB.vm'** to asm |
| **ClassA.0** will represent the **first** static symbol and will get the **first** place on RAM (16)  **ClassA.1** will represent the **second** static symbol and will get the **second** place on RAM (17)  // push static 0 from **'ClassA.vm'** in asm:  @ClassA.0 //@16  D=M  @SP  A=M  M=D  @SP  M=M+1  // push static 1 from **'ClassA.vm'** in asm:  @ClassA.1 //@17  D=M  @SP  A=M  M=D  @SP  M=M+1  // pop static 0 from **'ClassA.vm'** in asm:  @SP  M=M-1  A=M  D=M  @ClassA.0 // @16  M=D  // pop static 1 from **'ClassA.vm'** in asm:  @SP  M=M-1  A=M  D=M  @ClassA.1 // @17  M=D | **ClassB.0** will represent the **third** static symbol and will get the **first** place on RAM (18)  **ClassB.1** will represent the **fourth** static symbol and will get the **second** place on RAM (19)  // push static 0 from **'ClassB.vm'** in asm:  @ClassB.0 //@18  D=M  @SP  A=M  M=D  @SP  M=M+1  // push static 1 from **'ClassB.vm'** in asm:  @ClassB.1 //@19  D=M  @SP  A=M  M=D  @SP  M=M+1  // pop static 0 from **'ClassB.vm'** in asm:  @SP  M=M-1  A=M  D=M  @ClassB.0 // @18  M=D  // pop static 1 from **'ClassB.vm'** in asm:  @SP  M=M-1  A=M  D=M  @ClassB.1 // @19  M=D |