Boaz Kantor
Introduction to Computer Science,
Fall semester 2009-2010
IDC Herzliya

Computer Architecture

"I know what you're thinking, 'cause right now I'm thinking the same thing.
Actually, I've been thinking it ever since I got here:
Why oh why didn't I take the BLUE pill? ". – Cypher, *The Matrix*

© Boaz Kantor, IDC

VIC – why oh why?

- If you want to know cars, it’s not enough to learn how to drive.
- VIC is like “under the hood” of computers.
- Why “like”? Because there’s no such language.
- But it simulates pretty damn good real computers architecture.

© Boaz Kantor, IDC

Reading input

- Each program has a queue of data.
- Every 'read' command reads the next number in the queue, and stores it in the data register.
- Syntax:
  - 800
- Example:
  - 800
  - 800
  - 800

© Boaz Kantor, IDC

Printing output

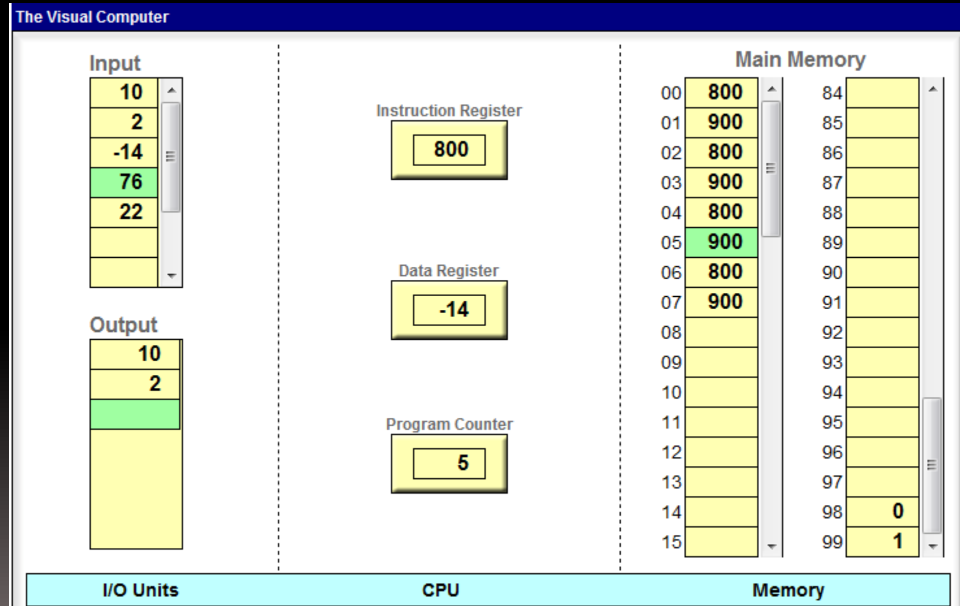
- VIC's output cell emulates a CMD window, a printer, a file or any other output device.
- We can print only what's in the data register.
- Remember: always make sure the data you want to print is in the data register!
- Syntax:
  - 900
- Example:
  - 900

© Boaz Kantor, IDC

I/O, example

- Exercise: read 4 numbers and write them to the output.
- Solutions:

```
800
900
800
900
800
900
800
900
```



© Boaz Kantor, IDC

Storing data

- Since there is only one data register, each ‘read’ instruction overwrites the data that was previously stored there.
- In order to keep data for future use, we store it in longer term memory.
- This memory is an array of cells.
- We can choose in which memory cell to store the data.
- Syntax:
  - `4xx`
- Example:
  - `481`
  - `482`

© Boaz Kantor, IDC

Reading and storing, example

Exercise

- Read 4 numbers
- Store them in memory cells 11, 12, 13, and 50.

Solution

```
800    // read number
411    // store in cell 11
800    // read number
412    // store in cell 12
800    // read number
413    // store in cell 13
800    // read number
450    // store in cell 50
```

© Boaz Kantor, IDC

Arithmetic operations

- The only processing available in VIC is adding and subtracting numbers.
- We add/subtract a number in a selected memory cell to/from the value in the data register.
- The result is saved back in the data register.
- Syntax:
  - `Add: 1xx`
  - `Subtract: 2xx`
- Example: add the number in cell 51 to the value in the data register:
  - `151`
- Example: subtract the value in cell 80 from the value in the data register:
  - `280`

© Boaz Kantor, IDC

Arithmetic operations, example

Exercise

- Read 4 numbers
- Sum up all the numbers
- Print the result

How many memory cells have we used? Can we use less memory to solve this exercise?

Solution

```
800        // read first number
490        // store in cell 90
800        // read second number
491        // store in cell 91
800        // read third number
492        // store in cell 92
800        // read fourth number
190        // add first number
191        // add second number
192        // add third number
900        // write to output
```

© Boaz Kantor, IDC

Memory efficiency, example

Naïve solution

```
800
490
800
491
800
492
800
190
191
192
900
```

Memory-efficient solution

```
800        // read first number
490        // store in cell 90
800        // read second number
190        // add first number
490        // store result
800        // read third number
190        // add previous result
490        // store result
800        // read fourth number
190        // add previous result
900        // write
```

© Boaz Kantor, IDC

Loops

- There is no ‘while’, ‘do-while’ nor ‘for’.
- We can only jump to another place in memory.
- Let’s ‘jump’ from this material and talk about something else. We will jump right back afterwards.

© Boaz Kantor, IDC

How it really works

- When we run a program, the operating system first loads the program instructions into the memory (RAM).
- The instructions are then read one by one (‘fetch’).
- The processor tries to understand what we wanted (‘decode’), and then runs the instruction (‘execute’).
- When we have a loop in the code, the instruction will be “jump to another location in memory and continue from there”.
- Data is saved in another location in the same memory (RAM).
- We can’t jump to the data area, and we can’t save data in the program area.

© Boaz Kantor, IDC

How it works with VIC

- When we load a '.vic' file to the VIC simulator, the program **instructions** are written to the main memory.
- Each instruction occupies one memory cell.
- We can use this to jump from one cell to another.
- VIC allows us to jump anywhere we want in the main memory (but we shouldn't).

© Boaz Kantor, IDC

Back to loops

- There is no ‘while’, ‘do-while’ nor ‘for’.
- We can only jump to another place in memory.
- There are 3 kinds of ‘jump’:
  - 5xx // go to cell xx
  - 6xx // if (data register == 0) go to cell xx
  - 7xx // if (data register > 0) go to cell xx
- Example:
  - 693

© Boaz Kantor, IDC

Loops, example

Exercise

- Write a program that reads 2 numbers and multiplies them.
- Let's think: 5x4 is 5+5+5+5.
- So we need these variables:
  - The number we multiply
  - Loop counter
  - Interim summation
- Let's set memory cells 90 for the number we multiply, 91 for the loop counter and 92 for the summation

Solution

Problem: Uninitialized!

```
398        // load zero
492        // initialize sum
800        // read the first number
490        // store as the number we summarize
800        // read the second number
491        // store as loop counter
390        // load the number
192        // add to the sum
492        // store the result
391        // load the loop counter
299        // loop-counter--
491        // store the new loop counter
704 706    // if (loop-counter>0) loop
392        // load the final summation
900        // write to output
```

© Boaz Kantor, IDC

Downsides of the 3-digit VIC code

- We have to know cell numbers **in advance**:
  - Branching: how do we know to which cell to jump?
  - Variables: how do we know we're not overriding the program area?
- Readability: it's hard to remember instructions as numbers.
- The solution: a higher level programming language!!

© Boaz Kantor, IDC

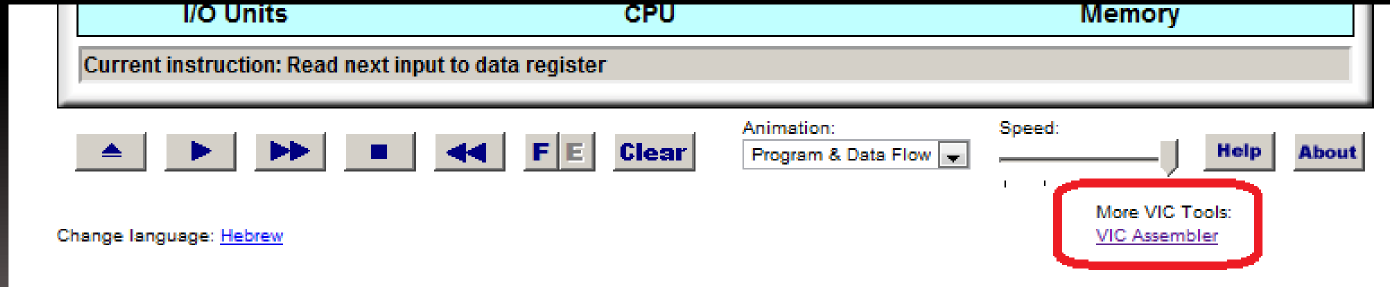
The symbolic VIC code

- It's a higher level programming language:
  - We use symbols instead of memory addresses.
  - The instructions are in English.
  - The source code is not executable. It needs to be translated into the 3-digit VIC code.
- The translation is called.. Compilation!

© Boaz Kantor, IDC

The VIC assembler

- Compiles the symbolic language to VIC code:
  - Translates English instructions to numbers.
  - Translates symbols to memory cells.



© Boaz Kantor, IDC

Loops, example (symbolic VIC)

3-digit VIC code

```
398        // load zero
492        // initialize summation
800        // read the first number
490        // store as the number we summarize
800        // read the second number
491        // store as loop counter
390        // load the number
192        // add to the summation
492        // store the result
391        // load the loop counter
299        // loop-counter--
491        // store the new loop counter
706        // if (loop-counter>0) loop
392        // load the final summation
900        // write to output
```

Symbolic

```
INIT:
        Load 98
        Store summation
        Read
        Store number
        Read
        Store counter
LOOP:
        Load number
        Add summation
        Store summation
        Load counter
        Sub 99
        Store counter
        Gotop LOOP
END:
        Load summation
        Write
```

© Boaz Kantor, IDC

If you can't see the font in class..

```
INIT:
        Load 98
        Store summation
        Read
        Store number
        Read
        Store counter
```

(continues to the right)

```
LOOP:
        Load number
        Add summation
        Store summation
        Load counter
        Sub 99
        Store counter
        Gotop LOOP
END:
        Load summation
        Write
```

© Boaz Kantor, IDC

Hacking (if we have time)

Code obfuscation

- Writing a program that writes itself:
  - Write an instruction which writes another instruction somewhere else in the code.

Buffer overflow

- Modify another program while running it:
  - Allocate space for user input.
  - Don’t check the length of the input.
  - The user input can be too long, while the overflowing characters are instructions, which will overwrite the instructions after the allocated memory.

© Boaz Kantor, IDC

Exercise #4, due Thursday, 17:00

- Learn and understand how VIC works.
- Learn and understand the differences between the 3-digit VIC code and the symbolic language.
- Make up a program and write it (add up a series of numbers).
- Identify where you have to write in VIC code and when in symbolic language.
- No styling conventions. Just be consistent. Keep labels on the left, indent the rest of the code once.
- Add comments to your symbolic programs.
- Good luck!

© Boaz Kantor, IDC