

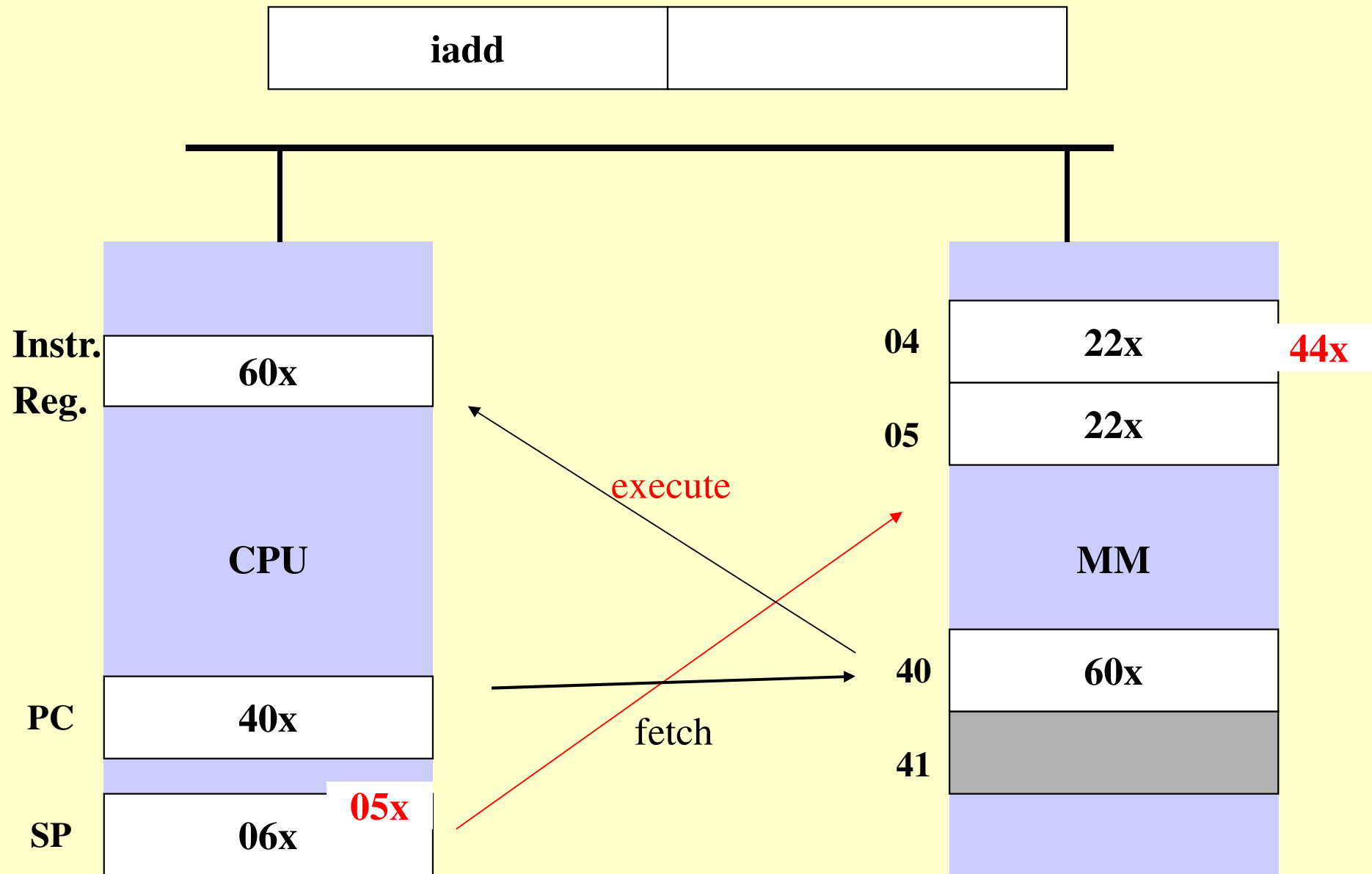
Graphical User Interfaces	Higher-level Programming
Operating Systems	
Low-level Programming	
Basic Machine Architecture	
Silicon	

CSCU9V4 Systems

Systems lecture 11

Computer Organisation

A Virtual Machine
a demonstration of V4 Virtual machine



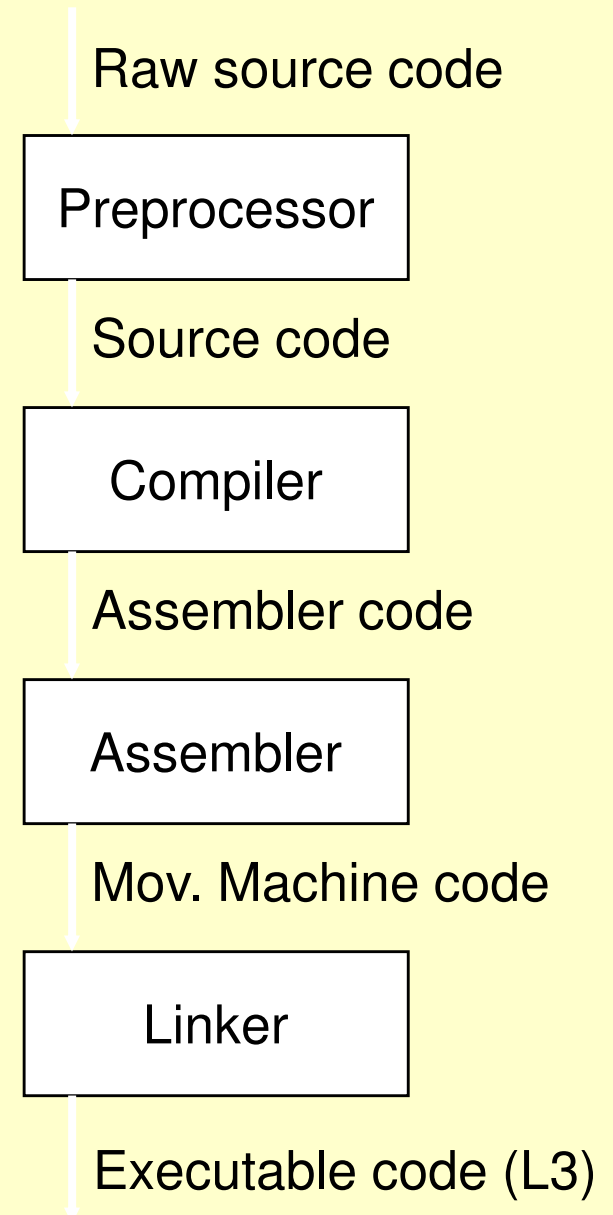
Most of the instructions (some more later)

Instruction (in hex)	Mnemonic	Translation
60ii	iadd	addition of top two stack elements (pops two numbers from the top of the stack, and pushes their sum back onto the stack)
64ii	isub	subtraction of top two stack elements
68ii	imul	multiplication of top two stack elements
6Cii	idiv	division of top two stack elements
70ii	irem	remainder operation on top two stack elements
7Eii	iand	boolean AND operation on top two stack elements
80ii	ior	boolean OR operation on top two stack elements
12xx	lcd #	pushes constant value xx onto top of stack
15xx	iload #	pushes a copy of memory location (localVariablesPointer+xx) onto top of stack
36xx	istore #	pops the top element of the stack and stores it in memory location (localVariablesPointer+xx)
a7xx	goto #	sets program counter to memory address xx
99xx	ifeq #	pops the top element of the stack, and if it equals zero, sets the program counter to xx

- ii: any two hex digits (ignored)
- xx: two hex digits used either as a constant (lcd) or address

An Example

- **High-level language:**
 - `some_var = 15 + 10`
- **Assembly language**
 - `ldc F`
 - `ldc A`
 - `iadd`
 - `istore local_var+0`
- **Movable machine code**
 - `0001 0010 00001111` * (push Fx onto stack)
 - `0001 0010 00001010` * (push Ax onto stack)
 - `0110 0000 00000000` * (add top 2 stack element)
 - `0011 0110 00000000` * (store at first local var)
- **Executable machine code (start at L=00001111)**
 - **40x** `0001 0010 00001111` * (push Ex onto stack)
 - **42x** `0001 0010 00001010` * (push Ax onto stack)
 - **44x** `0110 0000 00000000` * (add top 2 stack element)
 - **46x** `0011 0110 00000000` * (store at first local var)



the resulting GUI

V4 Virtual Machine

File Font

Local Variables: 04

Local Variables Pointer: 00

Stack Pointer: 04

Program Counter: 40

Instruction Register: 0000

Translation: NoOp

Execute Instruction

Thread Memory

00:	00000000	00	0
01:	00000000	00	0
02:	00000000	00	0
03:	00000000	00	0
04:	00000000	00	0
05:	00000000	00	0
06:	00000000	00	0
07:	00000000	00	0
08:	00000000	00	0
09:	00000000	00	0
0A:	00000000	00	0
0B:	00000000	00	0
0C:	00000000	00	0
0D:	00000000	00	0

Method Memory

40:	00000000	00	0
41:	00000000	00	0
42:	00000000	00	0
43:	00000000	00	0
44:	00000000	00	0
45:	00000000	00	0
46:	00000000	00	0
47:	00000000	00	0
48:	00000000	00	0
49:	00000000	00	0
4A:	00000000	00	0
4B:	00000000	00	0
4C:	00000000	00	0
4D:	00000000	00	0

Heap Memory

80:	00000000	00	0
81:	00000000	00	0
82:	00000000	00	0
83:	00000000	00	0
84:	00000000	00	0
85:	00000000	00	0
86:	00000000	00	0
87:	00000000	00	0
88:	00000000	00	0
89:	00000000	00	0
8A:	00000000	00	0
8B:	00000000	00	0
8C:	00000000	00	0
8D:	00000000	00	0

before we execute *iadd*

V4 Virtual Machine

File Font

Local Variables: 04

Local Variables Pointer: 00

Stack Pointer: 06

Program Counter: 40

Instruction Register: 6000

Translation: iadd

Execute Instruction

Thread Memory

00:	00000000	00	0
01:	00000000	00	0
02:	00000000	00	0
03:	00000000	00	0
04:	00010001	11	17
05:	00100010	22	34
06:	00000000	00	0
07:	00000000	00	0
08:	00000000	00	0
0B:	00000000	00	0
0C:	00000000	00	0

Method Memory

40:	01100000	60	96
41:	00000000	00	0
42:	00000000	00	0
43:	00000000	00	0
44:	00000000	00	0
45:	00000000	00	0
46:	00000000	00	0
47:	00000000	00	0
48:	00000000	00	0
49:	00000000	00	0
4D:	00000000	00	0

Heap Memory

80:	00000000	00	0
81:	00000000	00	0
82:	00000000	00	0
83:	00000000	00	0
87:	00000000	00	0
8B:	00000000	00	0
8C:	00000000	00	0
8D:	00000000	00	0

Entered manually

Entered manually

Entered manually

Press execute

Entered manually

Output filename:

Load Memory

Save Memory

Browse...

Browse...

after we execute *iadd*

V4 Virtual Machine

File Font

Local Variables: 04

Local Variables Pointer: 00

Stack Pointer: 05

Program Counter: 42

Instruction Register: 0000

Translation: NoOp

Execute Instruction

Thread Memory			Method Memory			Heap Memory		
00:	00000000	00 0	40:	01100000	60 96	80:	00000000	00 0
01:	00000000	00 0	41:	00000000	00 0	81:	00000000	00 0
02:	00000000	00 0	42:	00000000	00 0	82:	00000000	00 0
03:	00000000	00 0	43:	00000000	00 0	83:	00000000	00 0
04:	00110011	33 51	44:	00000000	00 0	84:	00000000	00 0
05:	00100010	22 34	45:	00000000	00 0	85:	00000000	00 0
06:	00000000	00 0	46:	00000000	00 0	86:	00000000	00 0
07:	00000000	00 0	47:	00000000	00 0	87:	00000000	00 0
08:	00000000	00 0	48:	00000000	00 0	88:	00000000	00 0
09:	00000000	00 0	49:	00000000	00 0	89:	00000000	00 0
0A:	00000000	00 0	4A:	00000000	00 0	8A:	00000000	00 0
0B:	00000000	00 0	4B:	00000000	00 0	8B:	00000000	00 0
0C:	00000000	00 0	4C:	00000000	00 0	8C:	00000000	00 0
0D:	00000000	00 0	4D:	00000000	00 0	8D:	00000000	00 0

Input filename:

Output filename:

Load Memory

Save Memory

Browse...

Browse...

answer

... but we put the data manually on the stack!

- **High-level language:**

- `some_var := 15 + 10`

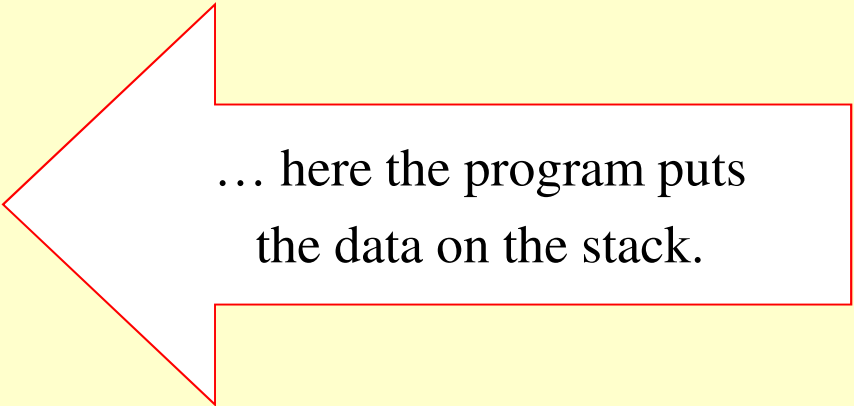
- **Assembly language**

- `ldc F`

- `ldc A`

- `iadd`

- `istore local_var+0`



... here the program puts
the data on the stack.

- **Movable machine code**

- `0001 0010 00001111` * (push Ex onto stack)

- `0001 0010 00001010` * (push Ax onto stack)

- `0110 0000 00000000` * (add top 2 stack element)

- `0011 0110 00000000` * (store at first local var)

- **Executable machine code (start at L=00001111)**

- **40x** `0001 0010 00001111` * (push Ex onto stack)

- **42x** `0001 0010 00001010` * (push Ax onto stack)

- **44x** `0110 0000 00000000` * (add top 2 stack element)

- **46x** `0011 0110 00000000` * (store at first local var)

At start

The screenshot shows the V4 Virtual Machine interface with the following components:

- Local Variables:** 04
- Local Variables Pointer:** 00
- Stack Pointer:** 04
- Program Counter:** 40
- Instruction Register:** 120F
- Translation:** ldc 0F
- Execute Instruction** button
- Thread Memory:** A table with 9 rows (00-08) and 3 columns. Rows 00-03 are highlighted in green.
- Method Memory:** A table with 13 rows (40-4A) and 3 columns. Row 40 is highlighted with a red arrow pointing to the Program Counter.
- Heap Memory:** A table with 7 rows (80-86) and 3 columns.

A yellow box with the text "Now press execute" is overlaid on the Heap Memory section. A red dotted arrow points from the "Execute Instruction" button to the "Translation" field.

- High-level language:
 - `some_var := 15 + 10`
- Assembly language
 - `ldc F`
 - `ldc A`
 - `iadd`
 - `istore local_var+0`

- Executable machine code (start at L=00001111)
 - `40x 0001 0010 00001111 * (push Fx onto stack)`
 - `42x 0001 0010 00001010 * (push Ax onto stack)`
 - `44x 0110 0000 00000000 * (add top 2 stack element)`
 - `46x 0011 0110 00000000 * (store at first local var)`

Executed 1st instruction

The screenshot shows the V4 Virtual Machine interface with the following components:

- Local Variables:** 04
- Local Variables Pointer:** 00
- Stack Pointer:** 05
- Program Counter:** 42
- Instruction Register:** 120A
- Translation:** ldc 0A
- Execute Instruction** button
- Thread Memory:** A table with 9 rows (00-08). Row 04 is highlighted in blue and contains the value 0F (15) in the third column.
- Method Memory:** A table with 13 rows (40-4A). Row 42 is highlighted in blue and contains the value 12 in the second column and 18 in the third column.
- Heap Memory:** A table with 7 rows (80-86). All values are 00 (0) in the third column.

Red dotted arrows indicate the flow of execution: from the **Execute Instruction** button to the **Instruction Register**, then to the **Translation** field, and finally to the **Method Memory** row 42. A yellow box with the text "Now press execute" is positioned over the **Heap Memory** table.

- High-level language:
 - `some_var := 15 + 10`
- Assembly language
 - `ldc F`
 - `ldc A`
 - `iadd`
 - `istore local_var+0`

- Executable machine code (start at L=00001111)
 - 40x 0001 0010 00001111 * (push Fx onto stack)
 - 42x 0001 0010 00001010 * (push Ax onto stack)
 - 44x 0110 0000 00000000 * (add top 2 stack element)
 - 46x 0011 0110 00000000 * (store at first local var)

Executed 2nd instruction

V4 Virtual Machine

File Font

Local Variables: 04

Local Variables Pointer: 00

Stack Pointer: 06

Program Counter: 44

Instruction Register: 6000

Translation: iadd

Execute Instruction

Thread Memory

00:	00000000	00	0
01:	00000000	00	0
02:	00000000	00	0
03:	00000000	00	0
04:	00001111	0F	15
05:	00001010	0A	10
06:	00000000	00	0
07:	00000000	00	0
08:	00000000	00	0

Method Memory

40:	00010010	12	18
41:	00001111	0F	15
42:	00010010	12	18
43:	00001010	0A	10
44:	01100000	60	96
45:	00000000	00	0
46:	00110110	36	54
47:	00000000	00	0
48:	00000000	00	0
49:	00000000	00	0
4A:	00000000	00	0

Heap Memory

80:	00000000	00	0
81:	00000000	00	0
82:	00000000	00	0
83:	00000000	00	0
84:	00000000	00	0
85:	00000000	00	0
86:	00000000	00	0
87:	00000000	00	0
88:	00000000	00	0
89:	00000000	00	0
8A:	00000000	00	0

Now press execute

- High-level language:
 - `some_var := 15 + 10`
- Assembly language
 - `ldc F`
 - `ldc A`
 - `iadd`
 - `istore local_var+0`
- Executable machine code (start at L=00001111)
 - `40x 0001 0010 00001111 * (push Fx onto stack)`
 - `42x 0001 0010 00001010 * (push Ax onto stack)`
 - `44x 0110 0000 00000000 * (add top 2 stack element)`
 - `46x 0011 0110 00000000 * (store at first local var)`

Executed 3rd instruction

The screenshot shows the V4 Virtual Machine interface with the following components:

- Local Variables:** 04
- Local Variables Pointer:** 00
- Stack Pointer:** 05
- Program Counter:** 46
- Instruction Register:** 3600
- Translation:** istore 00
- Execute Instruction** button
- Thread Memory:**

Address	Value	Label
00	00000000	00
01	00000000	00
02	00000000	00
03	00000000	00
04	00011001	19 25
05	00001010	0A 10
06	00000000	00 0
07	00000000	00 0
08	00000000	00 0
- Method Memory:**

Address	Value	Label
40	00010010	12 18
41	00001111	0F 15
42	00010010	12 18
43	00001010	0A 10
44	01100000	60 96
45	00000000	00 0
46	00110110	36 54
47	00000000	00 0
48	00000000	00 0
49	00000000	00 0
4A	00000000	00 0
- Heap Memory:**

Address	Value	Label
80	00000000	00 0
81	00000000	00 0
82	00000000	00 0
83	00000000	00 0
84	00000000	00 0
85	00000000	00 0
86	00000000	00 0

Annotations in the screenshot:

- A box labeled **sum** points to the value 25 at address 04 in Thread Memory.
- A yellow box labeled **Now press execute** points to the **Execute Instruction** button.
- Red dotted arrows trace the execution flow from the Instruction Register (3600) to the Method Memory (address 46) and then to the Thread Memory (address 04).

- High-level language:
 - `some_var := 15 + 10`
- Assembly language
 - `ldc F`
 - `ldc A`
 - `iadd`
 - `istore local_var+0`
- Executable machine code (start at L=00001111)
 - `40x 0001 0010 00001111 * (push Fx onto stack)`
 - `42x 0001 0010 00001010 * (push Ax onto stack)`
 - `44x 0110 0000 00000000 * (add top 2 stack element)`
 - `46x 0011 0110 00000000 * (store at first local var)`

Executed last instruction

The screenshot shows the V4 Virtual Machine interface. At the top, the title bar reads 'V4 Virtual Machine'. Below it are 'File' and 'Font' menus. The main area is divided into several sections:

- Local Variables:** 04 (green box)
- Local Variables Pointer:** 00 (green box)
- Stack Pointer:** 04 (blue box)
- Program Counter:** 48 (purple box)
- Instruction Register:** 0000 (purple box)
- Translation:** NoOp (purple box)
- Execute Instruction** button (grey box)

Below these are three memory panels:

- Thread Memory:** A table with 9 rows. The first row (00) is highlighted in green and contains the value 25. A box labeled 'sum' is placed over the first row. The second row (01) is highlighted in light green and contains 0. The third row (02) is highlighted in light blue and contains 0. The fourth row (03) is highlighted in light blue and contains 0. The fifth row (04) is highlighted in blue and contains 25. The sixth row (05) is highlighted in blue and contains 10. The seventh row (06) is highlighted in blue and contains 0. The eighth row (07) is highlighted in blue and contains 0. The ninth row (08) is highlighted in blue and contains 0.
- Method Memory:** A table with 13 rows. The first row (40) contains 12 and 18. The second row (41) contains 0F and 15. The third row (42) contains 12 and 18. The fourth row (43) contains 0A and 10. The fifth row (44) contains 60 and 96. The sixth row (45) contains 00 and 0. The seventh row (46) contains 36 and 54. The eighth row (47) contains 00 and 0. The ninth row (48) is highlighted in blue and contains 00 and 0. The tenth row (49) contains 00 and 0. The eleventh row (4A) contains 00 and 0.
- Heap Memory:** A table with 7 rows. The first row (80) contains 00 and 0. The second row (81) contains 00 and 0. The third row (82) contains 00 and 0. The fourth row (83) contains 00 and 0. The fifth row (84) contains 00 and 0. The sixth row (85) contains 00 and 0. The seventh row (86) contains 00 and 0.

A red dotted arrow points from the 'Execute Instruction' button to the instruction at address 46 in the Method Memory panel. A yellow box with the text 'has no effect' is placed over the instruction at address 48 in the Method Memory panel.

- **High-level language:**
 - `some_var := 15 + 10`
- **Assembly language**
 - `ldc F`
 - `ldc A`
 - `iadd`
 - `istore local_var+0`

- **Executable machine code (start at L=00001111)**
 - `40x 0001 0010 00001111 * (push Fx onto stack)`
 - `42x 0001 0010 00001010 * (push Ax onto stack)`
 - `44x 0110 0000 00000000 * (add top 2 stack element)`
 - `46x 0011 0110 00000000 * (store at first local var)`

Example program

40

1234 ldc 34 place constant (34x) on stack

12df ldc df place constant (dfX) on stack

6000 iadd add top two stack values

3602 istore 02 store value at location 2

- The first two hex digits define where the program will be loaded
- The remainder is the program
- Note that only the hex digits at the start of the line are used: the rest of the line is ignored.

Consider a program fragment ...

- Say we have a fragment of code

```
X = 2
```

```
if X != 0
```

```
    y = 4 ;
```

```
else y = 7 ;
```

How does this get translated into low-level language?

Translating

- What needs to be done?
 - assign locations to variables
X, y: Put X at location 0, and y at location 1
- Translate line by line.

```
X = 2      ildc 2 // push 2 on to stack
           istore 0 // store in location 0
```


Translating continued

```
If  X != 2          iload 0      // put x on stack
                                ildc 2      // put 2 on stack
                                sub      // subtract
                                ifeq ??     // jump to else part

y = 4                                ildc 4      // put 4 on
stack                                istore 1 // store it

else                                goto ?? // jump over else section

y = 7                                ildc 7      // put 7 on stack
                                istore 1 // store it
```

```
graph TD
    A["ifeq ?? // jump to else part"] --> B["else"]
    C["goto ?? // jump over else section"] --> D["y = 7"]
```

(end if)

And into machine code

```
40          // start address for loading
1202        // ildc 2 // push 2 on to stack
3600        //istore 0      // store in location 0
    1502 //iload 0 // put x on stack
    1202 //ildc 2  // put 2 on stack
    6400 //isub          // subtract
    99xx        // ifeq ??      // jump to else
part
```

- We don't know where the jump is to until we generates some more code.

Machine code cont'd

```
1204  //ildc 4           // put 4 on stack
3601           //istore 1 // store it
```

```
A7xx  //Goto xx jump to end of if statement
```

- Again, we don't know where to exactly

```
1207  //ildc 7           // put 7 on
stack
3601  //istore 1 // store it

0000  // do nothing
```

Resolving

```
40          // start address for loading
40 1202      // ildc 2  // push 2 on to stack
42 3600      //istore 0      // store in location 0
44      1502 //iload 2  // put x on stack
46      1202 //ildc 2  // put 2 on stack
48      6400 //isub      // subtract
4A      9952      // ifeq 0x52      // jump to
4C else part
4E      1204 //ildc 4  // put 4 on stack
50      3601      //istore 1      // store it
52      A756 //Goto 56 // jump to end of if
54 statement
56      1207 //ildc 7  // put 7 on stack
      3601 //istore 1      // store it
      0000 // do nothing
```

End of lecture