Smith College

Computer Science

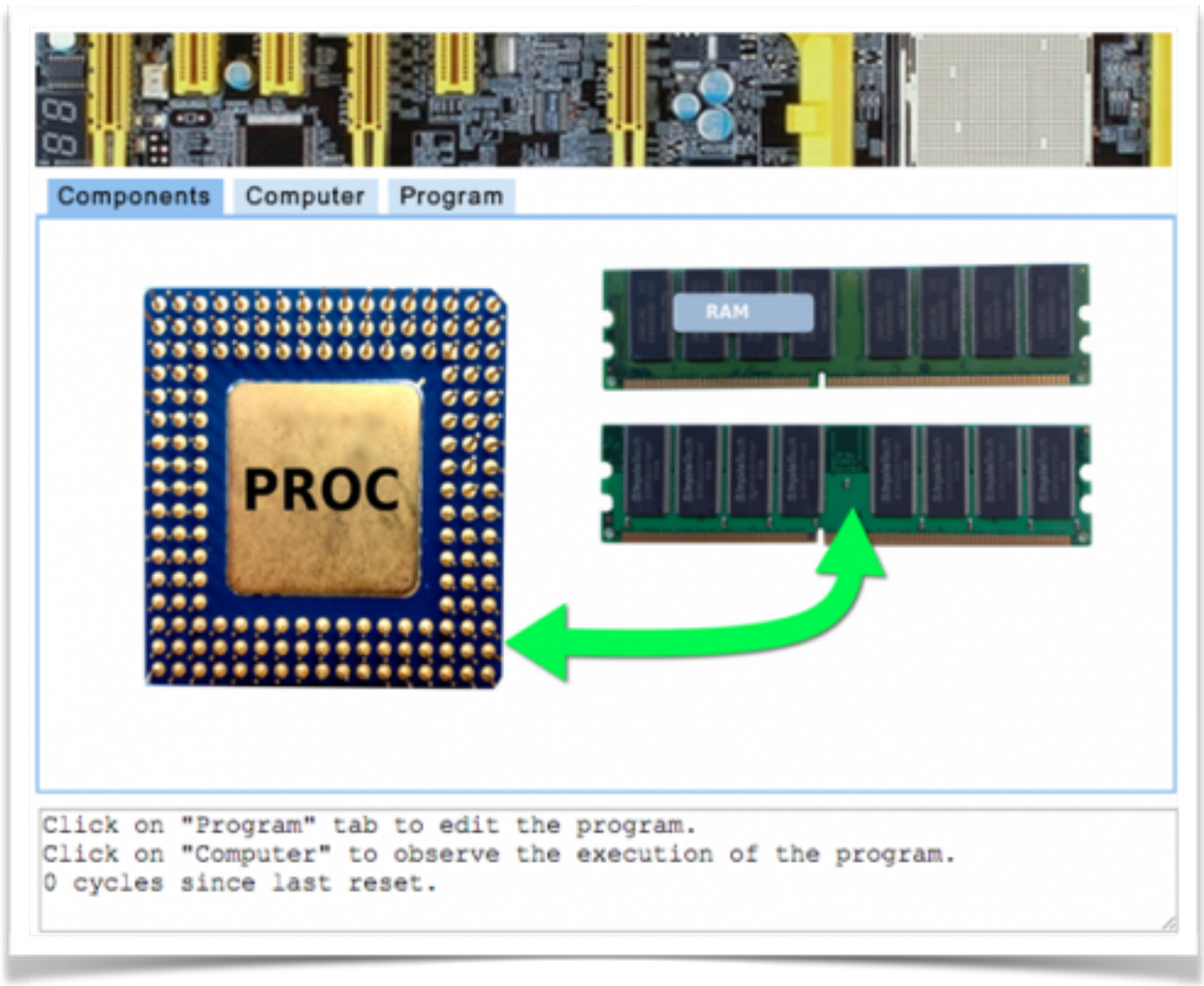
CSC231 — Assembly

Week #6 — Fall 2017

Dominique Thiébaut
dthiebaut@smith.edu

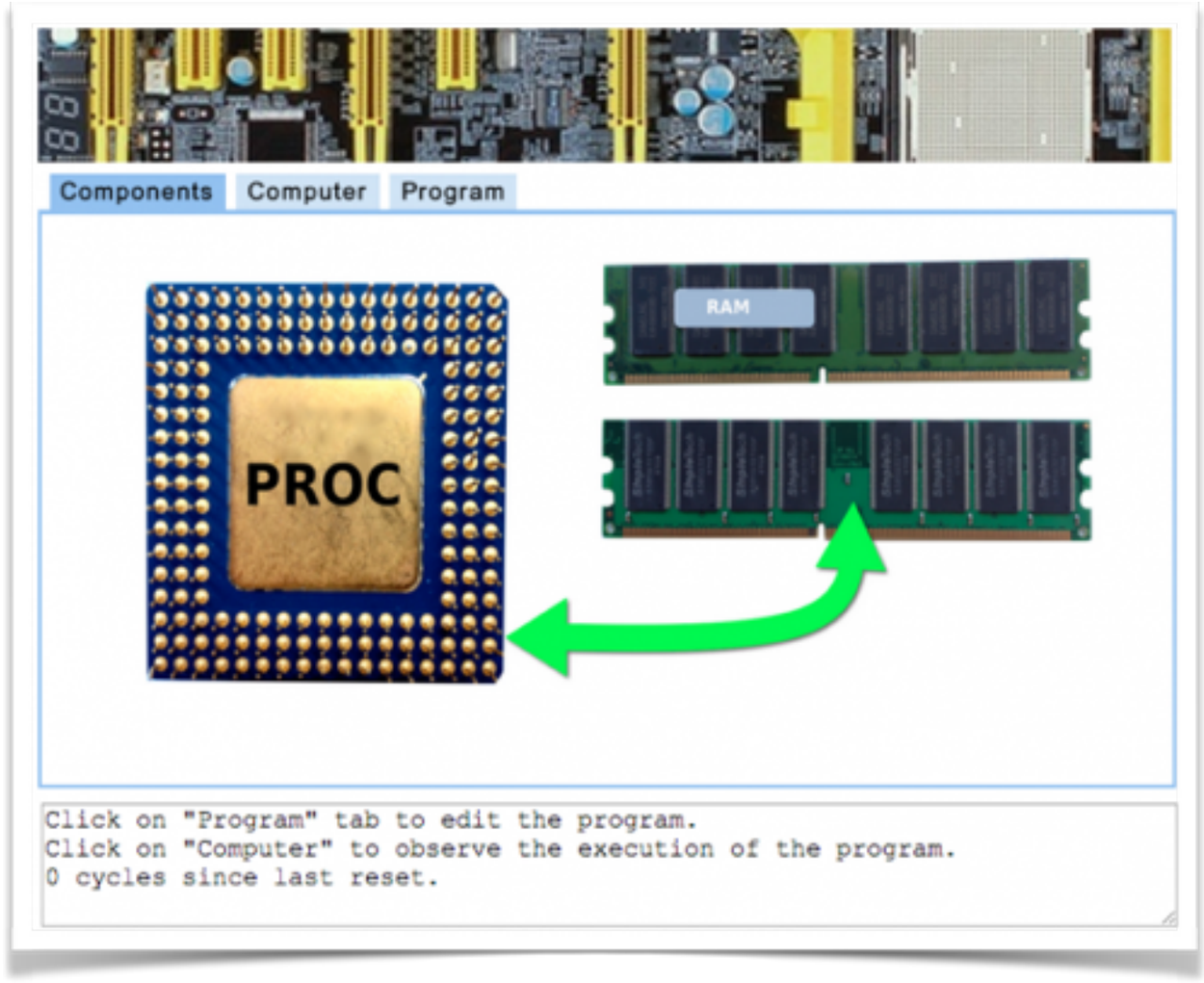
Computer Simulator

http://bit.ly/2jTJswI




```
0000:     LOAD [8]
0002:     ADD 1
0004:     STORE [8]
0006:     JUMP 0
0008:     0
```

http://www.science.smith.edu/dftwiki/media/simulator/

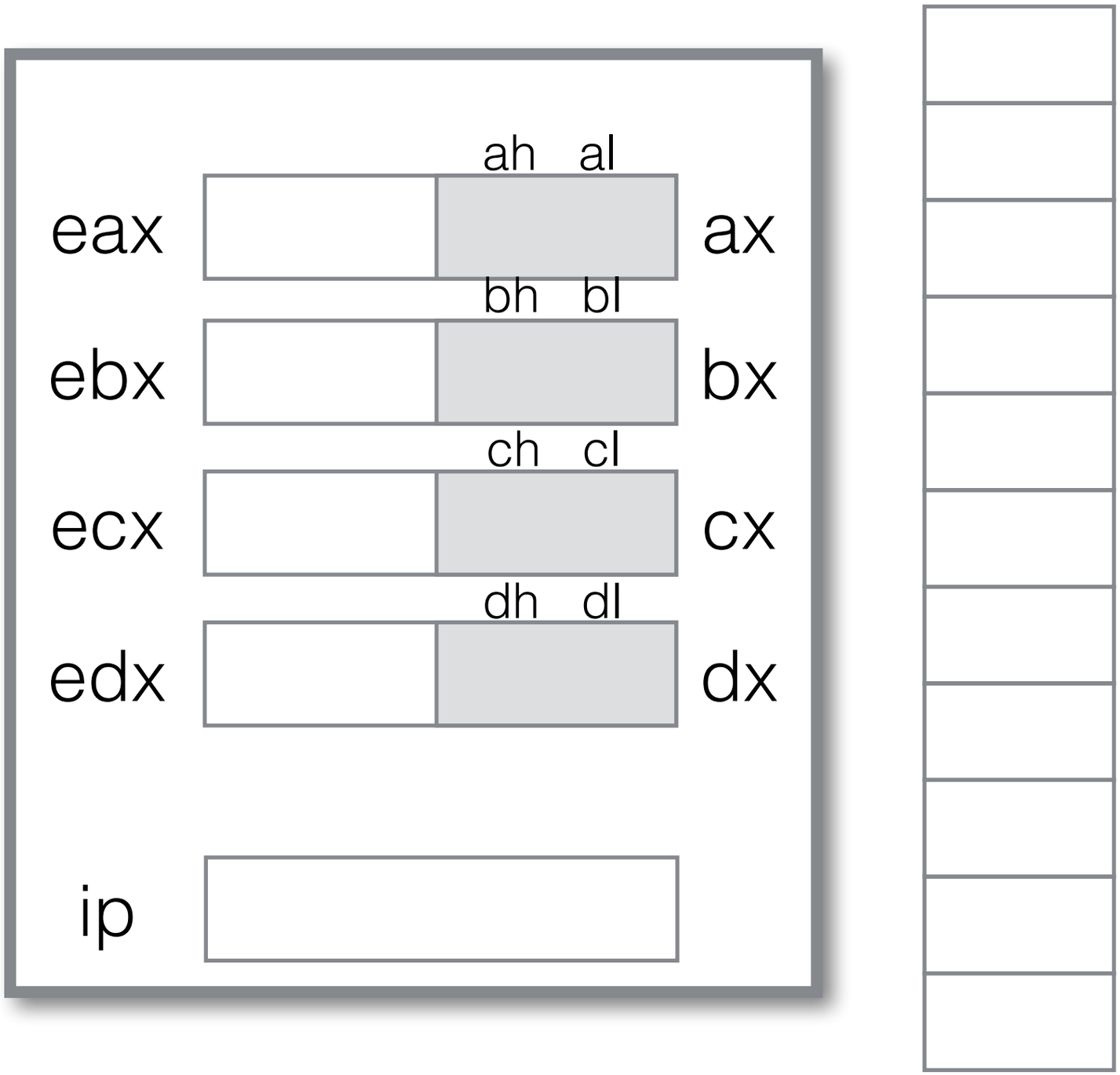


Resources

- Link to the simulator
- Link to the documentation
- Link to the CSC103 lab

IP Register

- The **IP** register (Instruction Pointer) in the Pentium plays the role of **PC** in the **Simple Computer Simulator**



Exercise

Compute x = (2*alpha + 1) / beta

```
alpha dd     3
beta  dd     4
x     dd     6
```

Exercise

Compute x = (2*alpha + 1) **%** beta

```
alpha dd     3
beta  dd     4
x     dd     6
```

```
n       db      231
```

Exercise

Print **n** in decimal.
Assume the 231Lib
library is **not** available…

A Solution

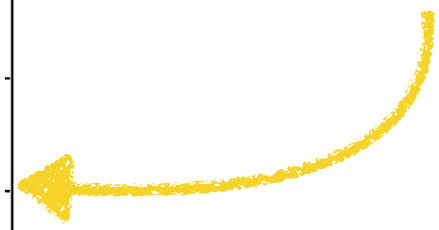
```
;;; printByteDec.asm
;;; D. Thiebaut
;;;
;;; Takes the byte stored in n and prints it in decimal
;;; on the screen.
;;; For example, if n is set to 231, the output of the program is
;;;
;;; n = 231
;;;
        section .data
n       db      231             ;the number to print in decimal
msg     db      "n = "          ;a message used to prefix the output
d1      db      ' '             ;the 3 digits composing n (some might
d2      db      ' '             ;end up being 0)
d3      db      ' '
lf      db      10              ;a line feed to terminate the printout

        global  _start
        section .text

_start: mov     eax, 0          ;use dwords for division
        mov     edx, 0          ;setup edx:eax <-- 0

        mov     al,  byte[n]    ;edx:eax <-- n
        mov     ebx, 10         ;get ready to divide by 10
        div     ebx             ;edx <- d3 in binary
                                ;eax <- quotient (d1d2)
        add     dl, '0'         ;binary to ascii
        mov     byte[d3], dl    ;d3 now contains ascii of
                                ;least significant digit of n

        mov     edx, 0          ;reset edx to 0
        div     ebx             ;edx <- d2 in binary
                                ;eax <- quotient (d1)
        add     dl, '0'         ;binary to ascii
        mov     byte[d2], dl    ;d2 now contains middle digit
        add     al, '0'         ;binary to ascii
        mov     byte[d1], al    ;d1 now contains most sig. digit

        mov     eax, 4          ;print "msg = d1d2d3" followed by
        mov     ebx, 1          ;line-feed
        mov     ecx, msg
        mov     edx, 4+4
        int     0x80

;;; exit
        mov     eax,1
        mov     ebx,0
        int     0x80    ; final system call
```

Logical Instructions

AND, OR, NOT, XOR

a	b	a and b
F	F	F
F	T	F
T	F	F
T	T	T

Truth Table



a	b	a and b
F	F	F
F	T	F
T	F	F
T	T	T

a	b	a and b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a and b
F	F	F
F	T	F
T	F	F
T	T	T

a	b	a and b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a or b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a and b
F	F	F
F	T	F
T	F	F
T	T	T

a	b	a and b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a or b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a xor b
0	0	0
0	1	1
1	0	1
1	1	0

a	b	a and b
F	F	F
F	T	F
T	F	F
T	T	T

a	b	a and b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a or b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	a xor b
0	0	0
0	1	1
1	0	1
1	1	0

a	not a
0	1
1	0

```
    10010
and 11100
---------
```

```
    10010
 or 11100
---------
```

```
    10010
xor 11100
---------
```

```
not 11100
---------
```

Instruction	Feature
AND	Good for setting bits to 0
OR	Good for setting bits to 1
XOR	Good for flipping bits
NOT	Good for complementing all the bits

Image credits http://www.staugustinechico.com/wp-content/uploads/2015/05/remember.jpg

and

and dest, src

```
and     op8,op8
and     op16,op16
and     op32,op32

op: mem, reg, imm
```

```
alpha db    0xf3
beta  dw    4
x     dd    6
```

```
and   byte[alpha],7
mov   ax,0x1234
and   ax,0xFF00

and   dword[x], 2
```

or

or dest, src

```
or      op8,op8
or      op16,op16
or      op32,op32

op: mem, reg, imm
```

```
alpha db    3
beta  dw    4
x     dw    0x0F06
```

```
or    byte[alpha],4
mov   ax,0x1234
or    ax,0xFF00

or    word[x], 15
```

xor

xor dest, src

```
xor      op8,op8
xor      op16,op16
xor      op32,op32

op: mem, reg, imm
```

```
alpha db     3
beta  dw     4
x     dd     0xF06
```

```
xor     byte[alpha],0x0F
mov     ax,0x1234
xor     ax,0xFF00

xor     dword[x], 15
```

not

not oprnd

```
not     op8
not     op16
not     op32

op: mem, reg
```

```
alpha db    3
beta  dw    4
x     dd    0xF06
```

```
not     byte[alpha]
mov     ax,0x1234
not     ax

not     dword[x]
```

Exercise

```
x       dd      0
most    dd      0
least   dd      0

        call    _getInput     ; eax <- user input
        mov     dword[x], eax

; set most to contain all 0s except most
; significant bit of x.  Set least to
; contain all 1s except least significant
; nybble of x
```

We stopped here last time...