Smith College

Computer Science

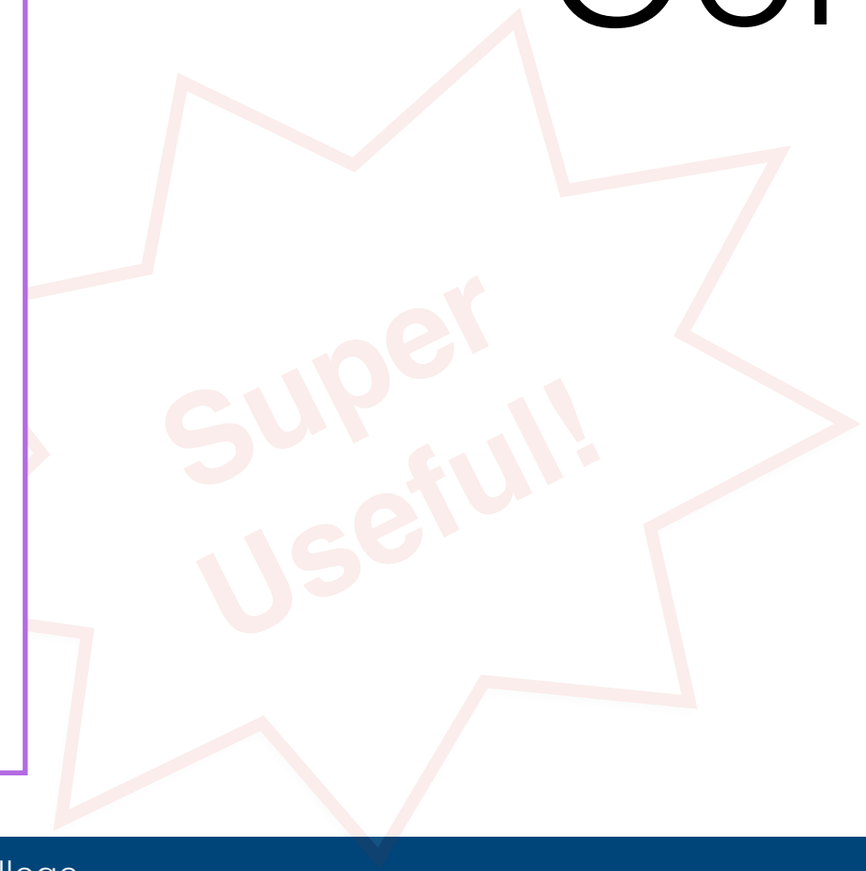
CSC231 — Assembly

Week #5 — Fall 2017

Dominique Thiébaut
dthiebaut@smith.edu

Conversion

0000	**0**	***0***
0001	**1**	***1***
0010	**2**	***2***
0011	**3**	***3***
0100	**4**	***4***
0101	**5**	***5***
0110	**6**	***6***
0111	**7**	***7***
1000	**8**	***8***
1001	**9**	***9***
1010	**A**	***10***
1011	**B**	***11***
1100	**C**	***12***
1101	**D**	***13***
1110	**E**	***14***
1111	**F**	***15***



Decimal to Binary Conversion

101001 =

$$101001 = 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$$

$$
\begin{aligned}
101001 &= 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0 \\
&= 32 + 0 + 8 + 0 + 0 + 1 \\
&= 41d
\end{aligned}
$$

$$
\begin{aligned}
101001 &= 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0 \\
&= 32 + 0 + 8 + 0 + 0 + 1 \\
&= 41d
\end{aligned}
$$

$$
\frac{101001}{2} = \frac{1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0}{2}
$$

$$101001 = 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$$
$$= 32 + 0 + 8 + 0 + 0 + 1$$
$$= 41d$$

$$\frac{101001}{2} = \frac{1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0}{2}$$

$$= \frac{1x2^5}{2} + \frac{0x2^4}{2} + \frac{1x2^3}{2} + \frac{0x2^2}{2} + \frac{0x2^1}{2} + \frac{1x2^0}{2}$$

$$101001 = 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$$
$$= 32 + 0 + 8 + 0 + 0 + 1$$
$$= 41d$$

$$\frac{101001}{2} = \frac{1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0}{2}$$

$$= \frac{1x2^5}{2} + \frac{0x2^4}{2} + \frac{1x2^3}{2} + \frac{0x2^2}{2} + \frac{0x2^1}{2} + \frac{1x2^0}{2}$$
$$= 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 0x2^0$$

$$101001 = 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$$
$$= 32 + 0 + 8 + 0 + 0 + 1$$
$$= 41d$$

$$\frac{101001}{2} = \frac{1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0}{2}$$

$$= \frac{1x2^5}{2} + \frac{0x2^4}{2} + \frac{1x2^3}{2} + \frac{0x2^2}{2} + \frac{0x2^1}{2} + \frac{1x2^0}{2}$$
$$= 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 0x2^0$$

$$= 10100$$

$$101001 = 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$$
$$= 32 + 0 + 8 + 0 + 0 + 1$$
$$= 41d$$

$$\frac{101001}{2} = \frac{1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0}{2}$$

$$= \frac{1x2^5}{2} + \frac{0x2^4}{2} + \frac{1x2^3}{2} + \frac{0x2^2}{2} + \frac{0x2^1}{2} + \frac{1x2^0}{2}$$
$$= 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 0x2^0$$

$$= 10100$$

$$101001 = 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$$
$$= 32 + 0 + 8 + 0 + 0 + 1$$
$$= 41d$$

$$\frac{101001}{2} = \frac{1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0}{2}$$

$$= \frac{1x2^5}{2} + \frac{0x2^4}{2} + \frac{1x2^3}{2} + \frac{0x2^2}{2} + \frac{0x2^1}{2} + \frac{1x2^0}{2}$$
$$= 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 0x2^0$$

$$= 10100$$
$$= 16 + 0 + 4 + 0 + 0$$
$$= 20d$$

Dividing by the base extracts the least significant digit

Python Program

```
# prompts user for an integer
# decomposes the integer into binary

x = int( input( "> " ) )
binary = ""

while True:
    if x==0:
        break

    remainder = x % 2
    quotient  = x // 2

    if remainder == 0:
        binary = "0" + binary
    else:
        binary = "1" + binary

    print( "%5d = %5d * 2 + %d    quotient=%5d remainder=%d binary=%16s"
           % (x, quotient, remainder, quotient, remainder, binary ) )

    x = quotient
```

Python Program

```
# prompts user for an int
# decomposes the integer

x = int( input( "> " ) )
binary = ""

while True:
    if x==0:
        break

    remainder = x % 2
    quotient  = x // 2

    if remainder == 0:
        binary = "0" + binary
    else:
        binary = "1" + binary

    print( "%5d = %5d * 2 + %d    quotient=%5d remainder=%d binary=%16s"
           % (x, quotient, remainder, quotient, remainder, binary ) )

    x = quotient
```

```
fiveNumbers35 — pi@raspberrypiREG: ~ — ssh — 88×20
pi@raspberrypiREG: ~                    pi@raspberrypiREG: ~                    Python
231b@aurora ~/handout $ ./decimal2binary.py
> 12345
12345 =  6172 * 2 + 1   (quotient= 6172 remainder=    1) -- binary=               1
 6172 =  3086 * 2 + 0   (quotient= 3086 remainder=    0) -- binary=              01
 3086 =  1543 * 2 + 0   (quotient= 1543 remainder=    0) -- binary=             001
 1543 =   771 * 2 + 1   (quotient=  771 remainder=    1) -- binary=            1001
  771 =   385 * 2 + 1   (quotient=  385 remainder=    1) -- binary=           11001
  385 =   192 * 2 + 1   (quotient=  192 remainder=    1) -- binary=          111001
  192 =    96 * 2 + 0   (quotient=   96 remainder=    0) -- binary=         0111001
   96 =    48 * 2 + 0   (quotient=   48 remainder=    0) -- binary=        00111001
   48 =    24 * 2 + 0   (quotient=   24 remainder=    0) -- binary=       000111001
   24 =    12 * 2 + 0   (quotient=   12 remainder=    0) -- binary=      0000111001
   12 =     6 * 2 + 0   (quotient=    6 remainder=    0) -- binary=     00000111001
    6 =     3 * 2 + 0   (quotient=    3 remainder=    0) -- binary=    000000111001
    3 =     1 * 2 + 1   (quotient=    1 remainder=    1) -- binary=   1000000111001
    1 =     0 * 2 + 1   (quotient=    0 remainder=    1) -- binary=  11000000111001
231b@aurora ~/handout $
```

Binary to Decimal

$$101001 = 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$$

$$= 32 + 0 + 8 + 0 + 0 + 1$$

$$= 41d$$

Binary to Hex

1010010 =

1010010 = 01010010

$$
\begin{aligned}
1010010 &= \quad 01010010 \\
&= \quad 0101\underset{\wedge}{}0010
\end{aligned}
$$

0000	**0**
0001	**1**
0010	**2**
0011	**3**
0100	**4**
0101	**5**
0110	**6**
0111	**7**
1000	**8**
1001	**9**
1010	**A**
1011	**B**
1100	**C**
1101	**D**
1110	**E**
1111	**F**

$$1010010 = 01010010$$
$$= 0101\,0010$$
$$\wedge$$

0000	**0**
0001	**1**
0010	**2**
0011	**3**
0100	**4**
0101	**5**
0110	**6**
0111	**7**
1000	**8**
1001	**9**
1010	**A**
1011	**B**
1100	**C**
1101	**D**
1110	**E**
1111	**F**

$$
\begin{aligned}
1010010 &= 01010010 \\
&= 0101\ 0010 \\
&= 5\quad 2
\end{aligned}
$$

Hex to Binary

0000	**0**
0001	**1**
0010	**2**
0011	**3**
0100	**4**
0101	**5**
0110	**6**
0111	**7**
1000	**8**
1001	**9**
1010	**A**
1011	**B**
1100	**C**
1101	**D**
1110	**E**
1111	**F**

5A1F =

0000	**0**
0001	**1**
0010	**2**
0011	**3**
0100	**4**
0101	**5**
0110	**6**
0111	**7**
1000	**8**
1001	**9**
1010	**A**
1011	**B**
1100	**C**
1101	**D**
1110	**E**
1111	**F**

5A1F = 0101 1010 0001 1111

Hex to Decimal

1A2F =

$$1A2F = 1x16^3 + Ax16^2 + 2x16^1 + Fx16^0$$

$$=$$

0000	**0**
0001	**1**
0010	**2**
0011	**3**
0100	**4**
0101	**5**
0110	**6**
0111	**7**
1000	**8**
1001	**9**
1010	**A**
1011	**B**
1100	**C**
1101	**D**
1110	**E**
1111	**F**

$$1A2F = 1x16^3 + Ax16^2 + 2x16^1 + Fx16^0$$
$$= 1x4096 + 10x256 + 2x16 + 15x1$$
$$=$$

0000	**0**
0001	**1**
0010	**2**
0011	**3**
0100	**4**
0101	**5**
0110	**6**
0111	**7**
1000	**8**
1001	**9**
1010	**A**
1011	**B**
1100	**C**
1101	**D**
1110	**E**
1111	**F**

$$1A2F = 1x16^3 + Ax16^2 + 2x16^1 + Fx16^0$$
$$= 1x4096 + 10x256 + 2x16 + 15x1$$
$$= 4096 + 2560 + 32 + 15$$
$$=$$

0000	**0**
0001	**1**
0010	**2**
0011	**3**
0100	**4**
0101	**5**
0110	**6**
0111	**7**
1000	**8**
1001	**9**
1010	**A**
1011	**B**
1100	**C**
1101	**D**
1110	**E**
1111	**F**

$$
\begin{aligned}
1A2F &= 1x16^3 + Ax16^2 + 2x16^1 + Fx16^0 \\
&= 1x4096 + 10x256 + 2x16 + 15x1 \\
&= 4096 + 2560 + 32 + 15 \\
&= 6703
\end{aligned}
$$

Decimal to Hex

Do:

Decimal —> Binary

Binary—> Hex

Exercises

http://www.science.smith.edu/dftwiki/index.php/CSC231_Review_of_hexadecimal_number_system

Convert these **hexadecimal** numbers to **binary**, and to **decimal**.

`1111`

`1234`

`AAAA`

`F001`

`FFFF`

Convert these **decimal** numbers to **binary**, and **hexadecimal**

65

127

What comes after these **hexadecimal** numbers, logically?

`aaAA`

`9999`

`19F`

`1ABF`

`FFEF`

`F00F`

`ABCDEF`

Perform the following additions in hex

```
  1000        1234        1234        FFFF
+ AAAA      +    F      + ABCD      + FFFF
```

More Arithmetic Instructions

Intel Pentium 4 Northwood

Buffer Allocation & Register Rename

Instruction Queue (for less critical fields of the uOps)

General Instruction Address Queue & Memory Instruction Address Queue (queues register entries and latency fields of the uOps for scheduling)

Floating Point, MMX, SSE2 Renamed Register File 128 entries of 128 bit.

uOp Schedulers

FP Move Scheduler: (8x8 dependency matrix)

Parallel (Matrix) Scheduler for the two double pumped ALU's

General Floating Point and Slow Integer Scheduler: (8x8 dependency matrix)

Load / Store uOp Scheduler: (8x8 dependency matrix)

Load / Store Linear Address Collision History Table

Integer Execution Core

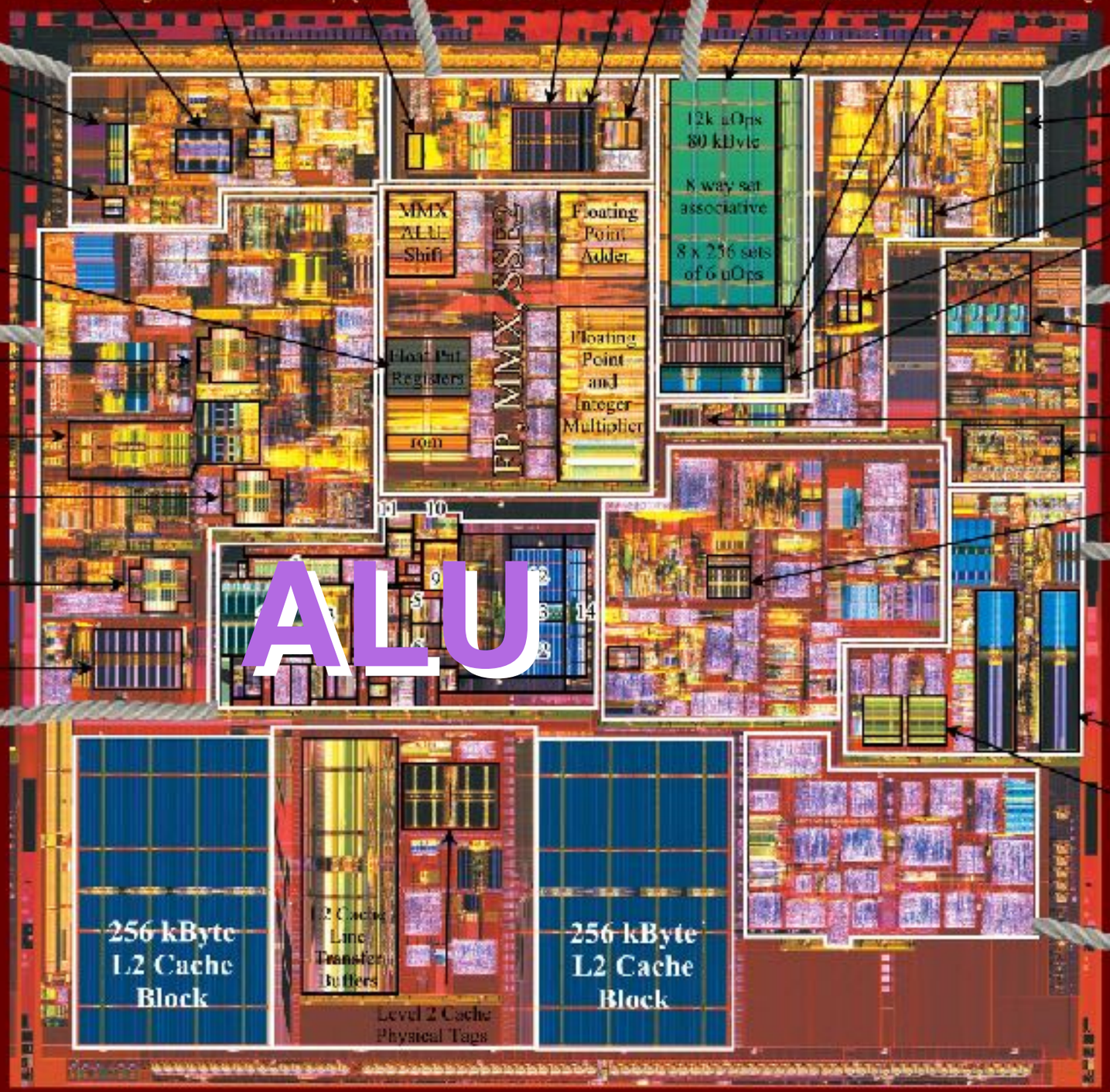
(1) uOp Dispatch unit & Replay Buffer Dispatches up to 6 uOps / cycle
(2) Integer Renamed Register File 128 entries of 32 bit + 6 status flags 12 read ports and six write ports
(3) Databus switch & Bypasses to and from the Integer Register File.
(4) Flags. Write Back
(5) Double Pumped ALU 0
(6) Double Pumped ALU 1
(7) Load Address Generator Unit
(8) Store Address Generator Unit
(9) Load Buffer (48 entries)
(10) Store Buffer (24 entries)

Execution Pipeline Start

Register Alias History Tables (2x126)
Register Alias Tables
uOp Queue

Instruction Trace Cache

Micro code Sequencer
Micro code ROM & Flash
Trace Cache Fill Buffers
Distributed Tag comparators 24 bit virtual Tags



(11) ROB Reorder Buffer 3x42 entries
(12) 8 kByte Level 1 Data cache four way set associative. 1R/1W
(13) Summed Address Index decode and Way Predict
(14) Cache Line Read / Write Transferbuffers and 256 bit wide bus to and from L2 cache

Trace Cache Access, next Address Predict

Trace Cache Branch Prediction Table (BTB), 512 entries.

Return Stacks (2x16 entries)

Trace Cache next IP's (2x)

Miscellaneous Tag Data

Instruction Decoder

Up to 4 decoded uOps/cycle on (from max. one x86 instr/cycle) Instructions with more than four are handled by Micro Sequencer

Trace Cache LRU bits

Raw Instruction Bytes in

Data TLB, 64 entry fully associative, between threads dual ported (for loads and stores)

Instruction Fetch from L2 cache and Branch Prediction

Front End Branch Prediction Tables (BTB), shared, 4096 entries in total

Instruction TLB's 2x64 entry, fully associative for 4k and 4M pages. In: Virtual address [31:12] Out: Physical address [35:12] + 2 page level bits

Front Side Bus Interface, 400..800 MHz

April 19, 2003 www.chip-architect.com

Right now,
we are dealing only
with **UNSIGNED** integers!

inc

inc operand

```
inc     reg8
inc     reg16
inc     reg32
inc     mem8
inc     mem16
inc     mem32
```

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

```
inc   al
inc   cx
inc   ebx

inc   word[beta]   ;beta <- 5
inc   dword[x]     ;x <- 1
```

dec

dec operand

```
dec     reg8
dec     reg16
dec     reg32
dec     mem8
dec     mem16
dec     mem32
```

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

```
dec   al              ;al <- al-1
dec   cx
dec   ebx

dec   word[beta]      ;beta <- 3
dec   dword[x]        ;x <- 5
```

mul

mul operand

```
mul     reg8
mul     reg16
mul     reg32
mul     mem8
mul     mem16
mul     mem32
```

Observation

```
   1001
x  1110
```

Observation

```
   1001
x  1110
-------
   0000
```

Observation

```
   1001
x  1110
-------
   0000
  1001
```

Observation

```
   1001
x  1110
-------
   0000
  1001
 1001
```

Observation

```
   1001
x  1110
-------
   0000
  1001
 1001
1001
-------
```

Observation

```
   1001
x  1110
-------
   0000
  1001
 1001
1001
-------
1111110
```

mul

mul operand

```
mul     reg8
mul     reg16
mul     reg32
mul     mem8
mul     mem16
mul     mem32
```

edx:eax <— operand$_{32}$ * eax
dx:ax <— operand$_{16}$ * ax
ax <— operand$_{8}$ * al

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

      mul   byte[alpha]   ;ax <- al*alpha
      mul   ebx           ;edx:eax <-
                          ;     ebx*eax
```

This has tremendously important **consequences**!

```
public class JavaLimits {

        public static void main(String[] args) {
               int a = 1;
               for ( int i = 0; i< 100; i++ ){
                    a = a * 2;
                    System.out.println( "a = " + a );
               }
        }
}
```

```
public class JavaLimits {

        public static void main(String[] args) {
                // ----------------------------------------
                // a multiplication of ints
                int x = 0x30000001;
                int y = 0x30000001;

                System.out.println( "x = " + x );
                System.out.println( "y = " + y );

                int z = x * y;

                System.out.println( "z = " + z );
                System.out.println();

        }
}
```

```
x = 805306369
y = 805306369
z = 1610612737
```

How big is a 32-bit **int**?

Ranges
(Unsigned Integers)

8 bits	0 - 255
16 bits	0 - 65,535
32 bits	0 - 4,294,967,295

div

```
div     reg8
div     reg16
div     reg32
div     mem8
div     mem16
div     mem32
```

div operand

R : Q	
edx:eax	<— edx:eax/ $operand_{32}$
dx:ax	<— dx:ax / $operand_{16}$
ah:al	<— ax / $operand_{8}$

```
alpha db    3
beta  dw    4
x     dd    6
;compute beta/alpha
      mov   ax, word[beta]
      div   byte[alpha]
;quotient in al
;remainder in ah
```

Exercise

Compute x = 2*alpha +3*beta + x - 1

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

End of Video Lecture
(Monday 10/2/17)

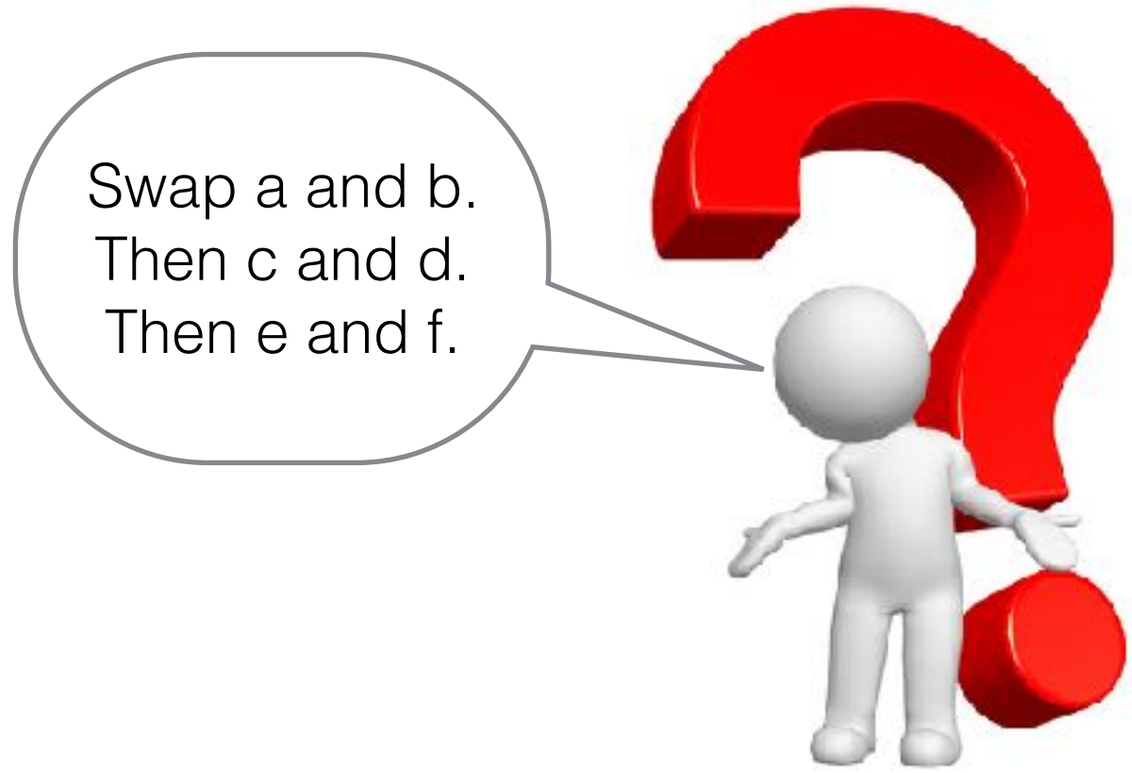
Elements of Style
Past-Due Homework 2

```
;;; hw2.asm
;;; Ha Cao
;;; The program takes inputs of a, b, c and computes
;;;      ans = 3*(a+b-c)+2(c-1)
;;;
;;; Assemble and run the program as follows:
;;; nasm -f elf hw2.asm
;;; nasm -f elf 231Lib.asm
;;; ld -melf_i386 -o hw2 hw2.o 231Lib.o

                extern  _printDec
                extern  _printInt
                extern  _printString
                extern  _println
                extern  _getInput

        section .data
prompt          db      "> "
promptLen       equ     $-prompt
ansStr          db      "ans = "
ansStrLen       equ     $-ansStr

a               dd      0
b               dd      0
c               dd      0
ans             dd      0

                section .text
                global  _start
_start:
        ;; display prompt
                mov     ecx, prompt
                mov     edx, promptLen
                call    _printString
        ;; get a
                call    _getInput
                mov     dword[a], eax

        ;; display prompt
                mov     ecx, prompt
                mov     edx, promptLen
                call    _printString
        ;; get b
                call    _getInput
                mov     dword[b], eax

        ;; display prompt
                mov     ecx, prompt
                mov     edx, promptLen
                call    _printString
```

```
        ;; get c
                call    _getInput
                mov     dword[c], eax

        ;; -----------------------------------
        ;; computation: ans = 3*(a+b-c) + 2*(c-1)
        ;; -----------------------------------

        ; your code will go here...
                mov     eax, 0          ; clear value left in
                                        ; registers
                mov     ebx, 0
                mov     ecx, 0
                mov     edx, 0

                add     eax, dword[a] ; add a and b into eax
                add     eax, dword[b]
                sub     eax, dword[c] ; subtract c from eax
                add     ecx, eax      ; add eax into ecx 3
                                      ; times
                add     ecx, eax
                add     ecx, ex
                add     ebx, dword[c] ; add c into ebx
                sub     ebx, 1        ; subtract 1 from ebx
                add     edx, ebx      ; add ebx into edx 2
                                      ; times
                add     edx, ebx
                add     ecx, edx      ; add edx to ecx
                mov     dword[ans], ecx ; mov the value in ecx
                                        ; into the memory at ans

                ;; -----------------------------------
                ;; display "ans = "
                ;; -----------------------------------
                        mov     ecx, ansStr
                        mov     edx, ansStrLen
                        call    _printString

                ;; -----------------------------------
                ;; display ans variable
                ;; -----------------------------------
                        mov     eax, dword[ans]
                        call    _printInt
                        call    _println
                        call    _println

;;; exit
                mov     ebx, 0
                mov     eax, 1
                int     0x80
```

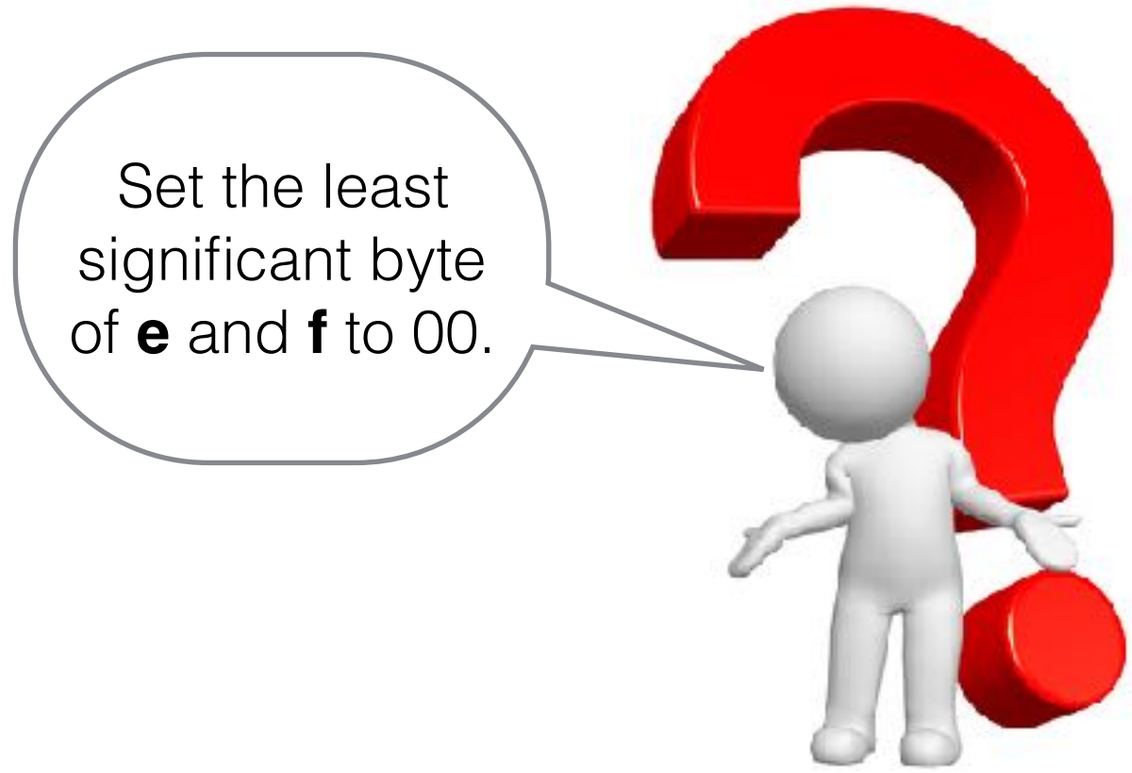
Exercise

```
            section .data
a           db      10
b           db      0
c           dw      0x1234
d           dw      0
e           dd      0
f           dd      0x12345678

            section .text
```



Exercise

```
            section .data
a           db      10
b           db      0
c           dw      0x1234
d           dw      0
e           dd      0xcdef
f           dd      0x12345678

            section .text
```



Exercise

```
            section .data
a           db
b           db
c           dw
d           dw
e           dd
f           db

            section .text
```

99
88
77
66
55
44
33
22
1f
1a
11
00

hex

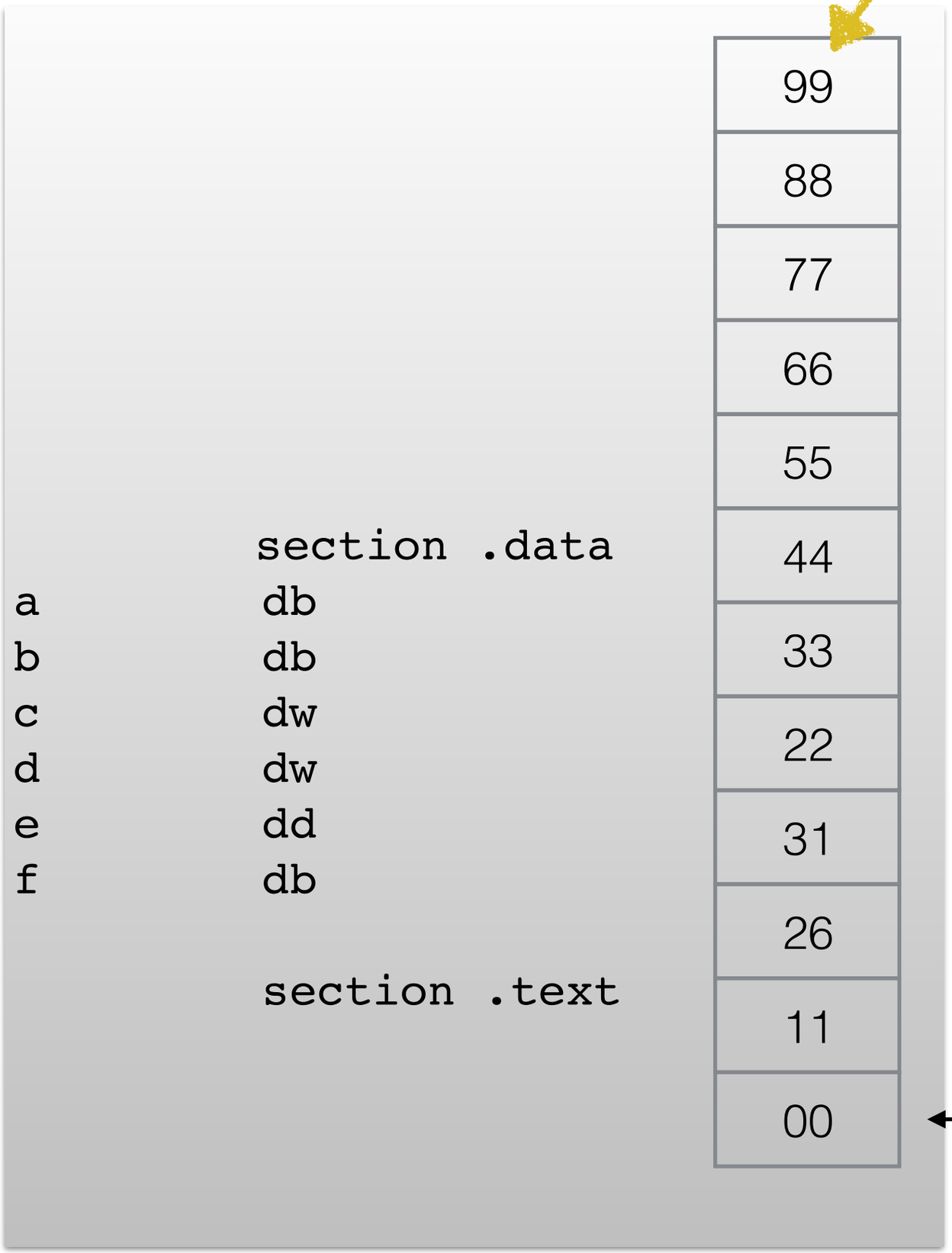
a

reconstruct the declarations of a, b, c, d, e and f.

typical midterm question!

Exercise

```
            section .data
a           db
b           db
c           dw
d           dw
e           dd
f           db

            section .text
```



reconstruct the declarations of a, b, c, d, e and f.

typical midterm question!

Simple Rule
Conversion from byte/word to dword

```
alpha    db    15
beta     dw    10000

; put alpha in eax

; put beta in ebx
```

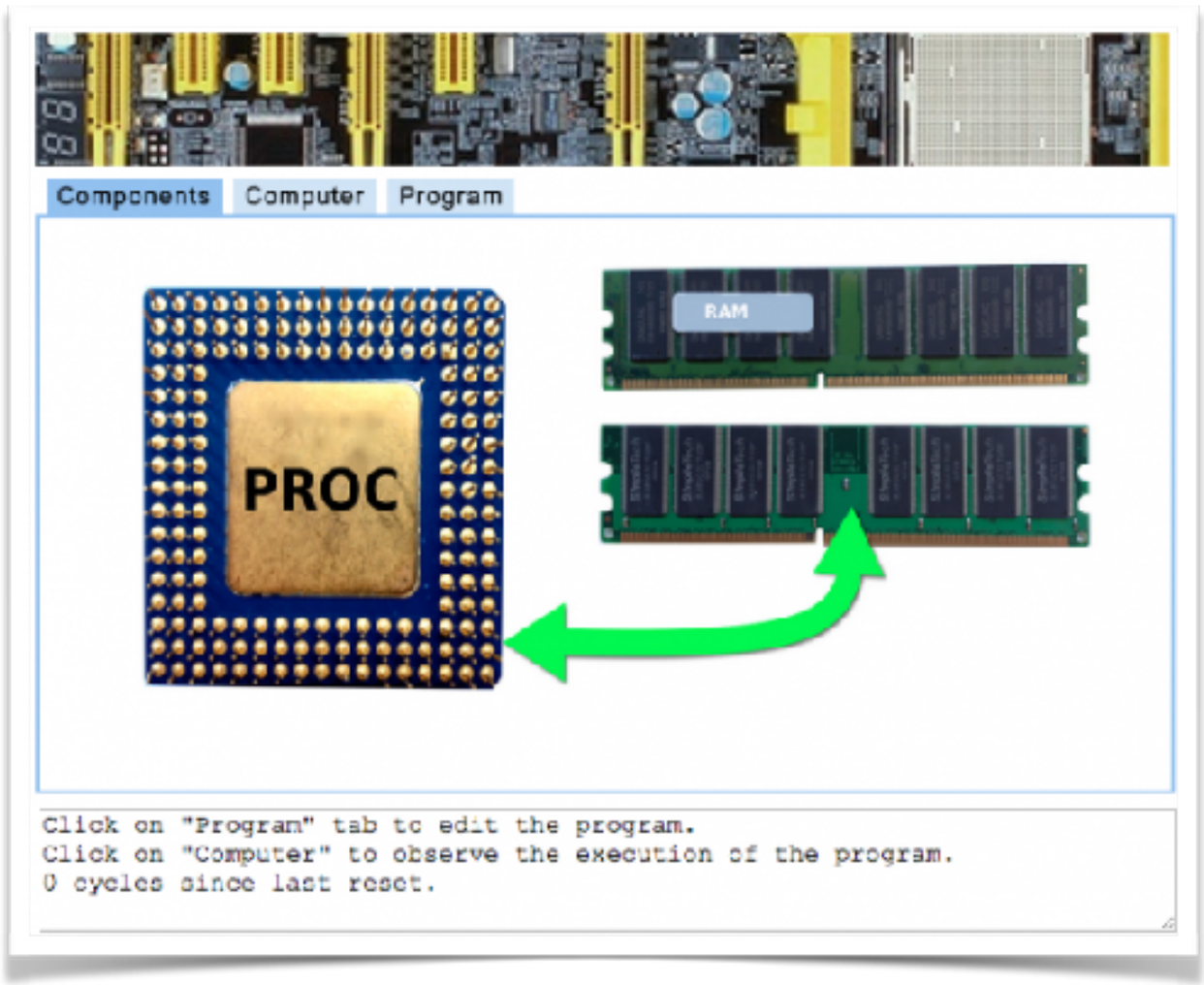
Simple Rule
Conversion from byte/word to dword


```
alpha    db    15
beta     dw    10000

; put alpha in eax

; put beta in ebx
```

Following the step by step execution of the program

Computer Simulator

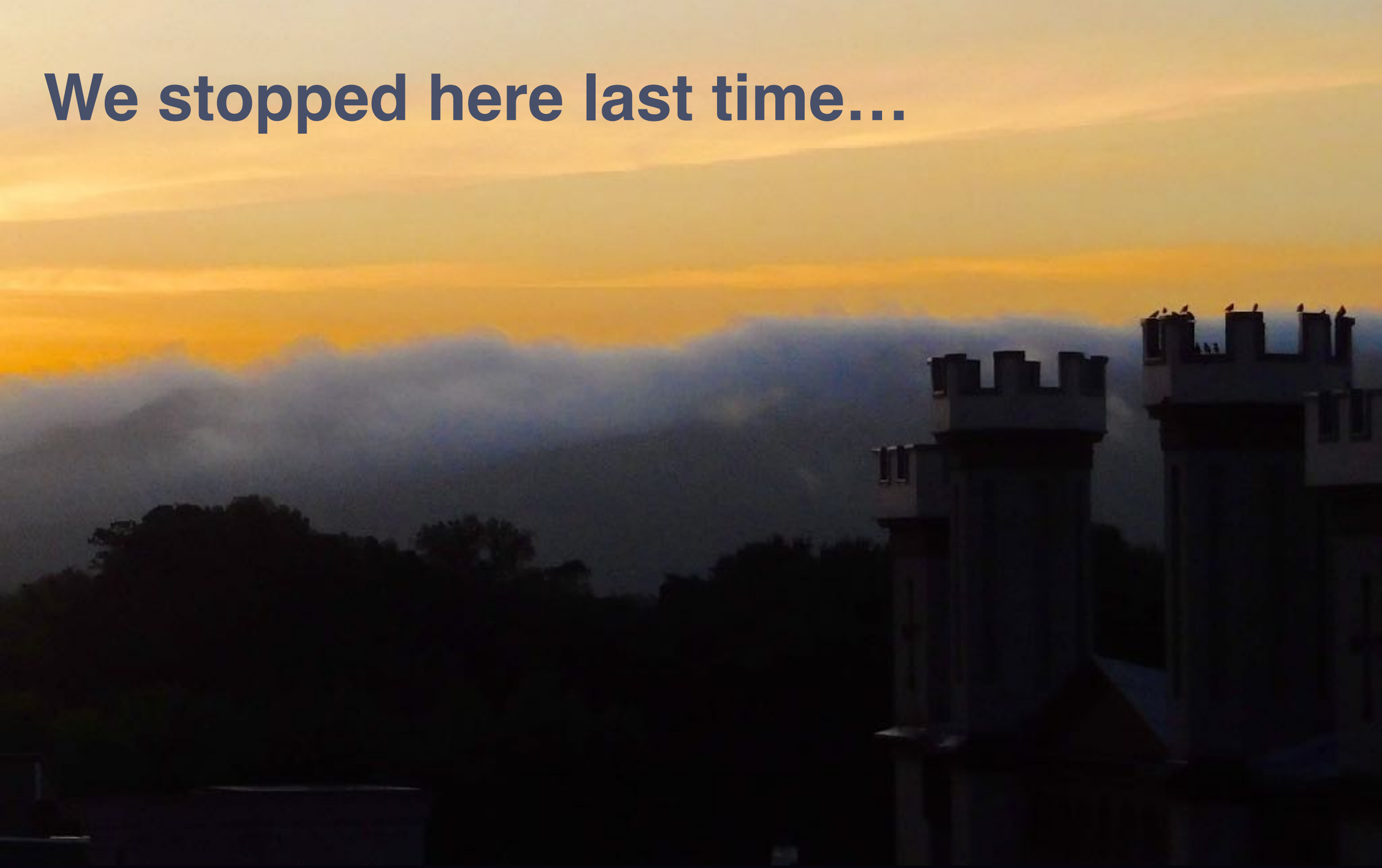
http://bit.ly/2jTJswI




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

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

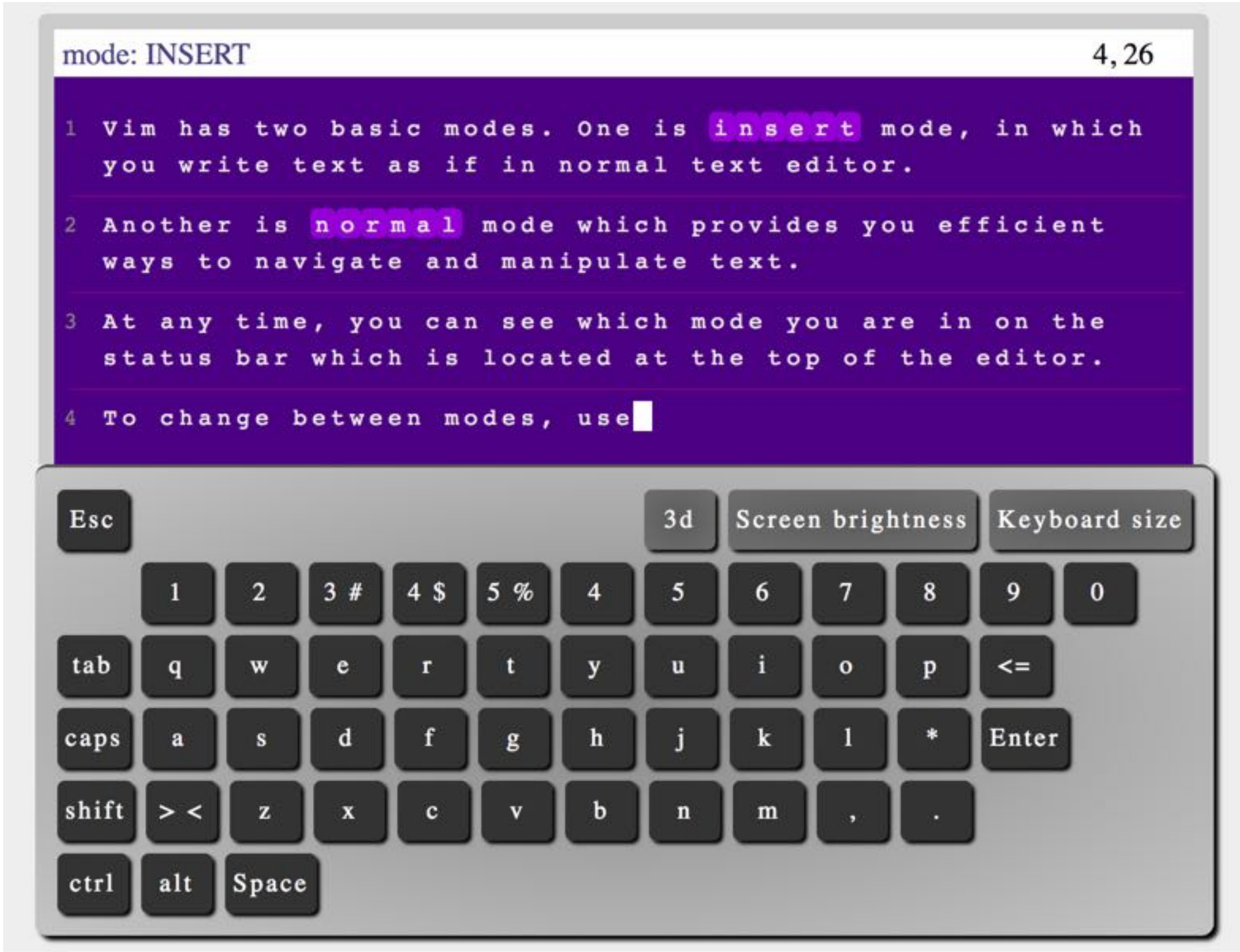
We stopped here last time…



Exercise

```
amount dd     1234
no20s  dd     0
no10s  dd     0
no5s   dd     0
no1s   dd     0

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

; break down amount into 20s, 10s, 5s and 1s
```

Vi & Redirection Lab



http://www.science.smith.edu/dftwiki/index.php/CSC231_Bash_Tutorial_4