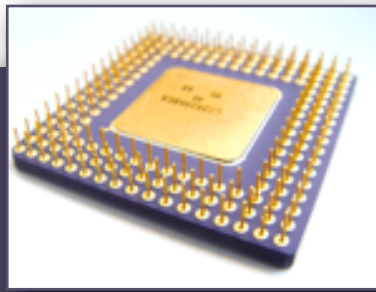


§3.5

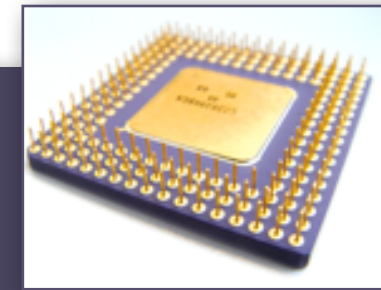
Symbolic Constants

Homework



- ▶ **Quiz 1** on 9/22 – one week from Monday; **Exam 1** TBA
 - ▶ **Homework 2** is due in one week – Friday, Sept 19, 11 a.m.
 - ▶ Submit electronically in Canvas
 - ▶ For next class (Monday, September 15):
 - ▶ Read about the **Assemble-Link-Execute Cycle** on p. 71 (skip the rest of §3.3)
 - ▶ What is a **linker**? An **object file**? Your book's description is not very good, so Google these terms.
 - ▶ Note that the linker copies procedures from *statically* linked libraries into the executable. It does not copy procedures from *dynamically* linked libraries (DLLs); they are loaded at runtime.
 - ▶ Read **Sections 3.4–3.5** (omit §§3.4.7 and 3.4.8 on QWORD and TBYTE)
 - ▶ Covered Wednesday and today; more details in book
- Makeup exams must be scheduled in advance. Makeup exams will not be given after the exam is given in class.

Last Time

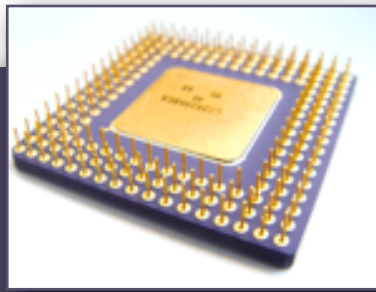


▶ §3.4 (Defining Data)

- ▶ BYTE, SBYTE, WORD, SWORD, DWORD, SDWORD, QWORD
- ▶ DUP operator
- ▶ ? initializer
- ▶ Little vs. big endian
- ▶ Difference between .DATA and .DATA? directives
- ▶ Using **mov** for memory-register data movement

Finish Activity 6 (#6)

Symbolic Constants



Activity 7 #1-2

- ▶ Give a name to a constant value using =

CR = **0Dh**

LF = **0Ah**

NUL = **00h**

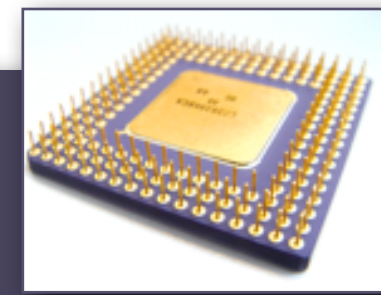
becomes

```
.data  
input BYTE "Hi", CR, LF, NUL
```

```
.data  
input BYTE "Hi", 0Dh, 0Ah, 00h
```

- ▶ Syntax: *name* = *expression* where *expression* is an integer constant or expression
- ▶ Read about EQU and TEXTEQU directives (§§3.5.3–3.5.4) – similar but different
- ▶ Symbolic constants are **not** stored in the resulting object file/executable
 - ▶ The assembler *replaces* them with their values *before* generating machine code
 - ▶ So the executable/machine code will be exactly the same as if you didn't use them

Current Location Counter (\$)



- ▶ \$ is a symbolic constant called the *current location counter*
- ▶ Its value is the memory address of the location at which it appears
- ▶ Note that the value of \$ depends on where it is written!

; Suppose the first declaration

; will be at offset 00405000h

.data

start = \$

value1 DWORD start

next = \$

value2 DWORD next

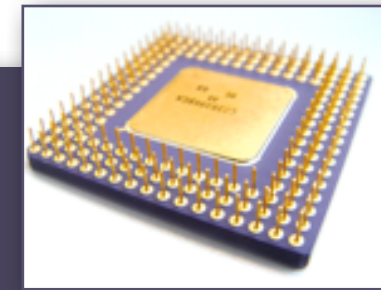
becomes

.data

value1 DWORD 00405000h

value2 DWORD 00405004h

Calculating the Size of a Byte Array

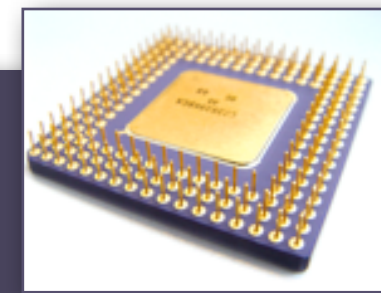


- ▶ **\$ is often used to determine the size of an array**
 - ▶ A label is just a name for a particular memory address
 - ▶ \$ is also a name for a memory address
 - ▶ Subtract to compute the number of bytes between the two

```
.data
hello BYTE "Hello", 0
len = ($-hello)

.code
mov eax, len
call WriteDec ; Prints 6
```

Calculating the Size of a Byte Array



- ▶ \$ is often used to determine the size of an array
- ▶ To be correct, **len = (\$-hello)** must appear immediately after the definition of hello. Why?

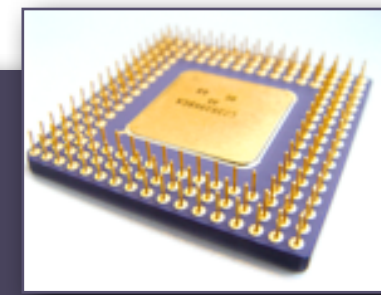
```
.data
hello BYTE "Hello", 0
len = ($-hello)

.code
mov eax, len
call WriteDec ; Prints 6
```

```
.data
hello BYTE "Hello", 0
moreBytes BYTE 0,0,0,0
len = ($-hello)

.code
mov eax, len
call WriteDec ; Prints 10
```

Calculating the Size of an Array



Activity 7 #3-5

- ▶ BYTE/SBYTE array: $(\$-start)$
- ▶ WORD/SWORD array: $(\$-start) / 2$
Equivalently, $(\$-start) / (SIZEOF \text{ WORD})$
- ▶ DWORD/SDWORD array: $(\$-start) / 4$
Equivalently, $(\$-start) / (SIZEOF \text{ DWORD})$

```
.data  
nums SWORD 1234h, 5678h, 9000h  
len = ($-nums) / (SIZEOF SWORD)
```

```
.code  
mov eax, len  
call WriteDec ; Prints 3
```

