Assembly Language

- Assembly language is a human-readable version of actual CPU instructions
- Ex – simple program to multiply two numbers:

C language program

```
unsigned char a, b, c;

void main()
{
a = 3;
b = 5;
c = a * b;
}
```

Assembly language program

```
           ; Multiplier Routine (8-bit x 8-bit = 16-bit product)
           ; ==================================================
           ; Shift and add algorithm
           ;

mult_8x8:  NAMEREG s0, multiplicand ; preserved
           NAMEREG s1, multiplier ; preserved
           NAMEREG s2, bit_mask ; modified
           NAMEREG s3, result_msb ; most-significant byte (MSB) of result,
           ; modified
           NAMEREG s4, result_lsb ; least-significant byte (LSB) of result,
           ; modified
           ;
           LOAD multiplicand, 05 ; 5 X 3
           LOAD multiplier, 03 ;
           LOAD bit_mask, 01 ; start with least-significant bit (lsb)
           LOAD result_msb, 00 ; clear product MSB
           LOAD result_lsb, 00 ; clear product LSB (not required)
           ;
           ; loop through all bits in multiplier
mult_loop: TEST multiplier, bit_mask ; check if bit is set
           JUMP Z, no_add ; if bit is not set, skip addition
           ;
           ADD result_msb, multiplicand ; addition only occurs in MSB
           ;
no_add:    SRA result_msb ; shift MSB right, CARRY into bit 7,
           ; lsb into CARRY
           SRA result_lsb ; shift LSB right,
           ; lsb from result_msb into bit 7
           ;
           SL0 bit_mask ; shift bit_mask left to examine
           ; next bit in multiplier
           ;
           JUMP NZ, mult_loop ; if all bit examined, then bit_mask = 0,
           ; loop if not 0
_end_main: JUMP _end_main; end of program!
```

Assembly Language vs. Machine Code

- The Hex representation of CPU instructions is often called machine code
- Machine Code is **NOT human readable**![*]

[*] At least not for the typical programmer…

Assembly language program

```
           ; Multiplier Routine (8-bit x 8-bit = 16-bit product)
           ; ==================================================
           ; Shift and add algorithm
           ;

mult_8x8:  NAMEREG s0, multiplicand ; preserved
           NAMEREG s1, multiplier ; preserved
           NAMEREG s2, bit_mask ; modified
           NAMEREG s3, result_msb ; most-significant byte (MSB) of result,
           ; modified
           NAMEREG s4, result_lsb ; least-significant byte (LSB) of result,
           ; modified
           ;
           LOAD multiplicand, 05 ; 5 X 3
           LOAD multiplier, 03 ;
           LOAD bit_mask, 01 ; start with least-significant bit (lsb)
           LOAD result_msb, 00 ; clear product MSB
           LOAD result_lsb, 00 ; clear product LSB (not required)
           ;
           ; loop through all bits in multiplier
mult_loop: TEST multiplier, bit_mask ; check if bit is set
           JUMP Z, no_add ; if bit is not set, skip addition
           ;
           ADD result_msb, multiplicand ; addition only occurs in MSB
           ;
no_add:    SRA result_msb ; shift MSB right, CARRY into bit 7,
           ; lsb into CARRY
           SRA result_lsb ; shift LSB right,
           ; lsb from result_msb into bit 7
           ;
           SL0 bit_mask ; shift bit_mask left to examine
           ; next bit in multiplier
           ;
           JUMP NZ, mult_loop ; if all bit examined, then bit_mask = 0,
           ; loop if not 0
_end_main: JUMP _end_main; end of program!
```

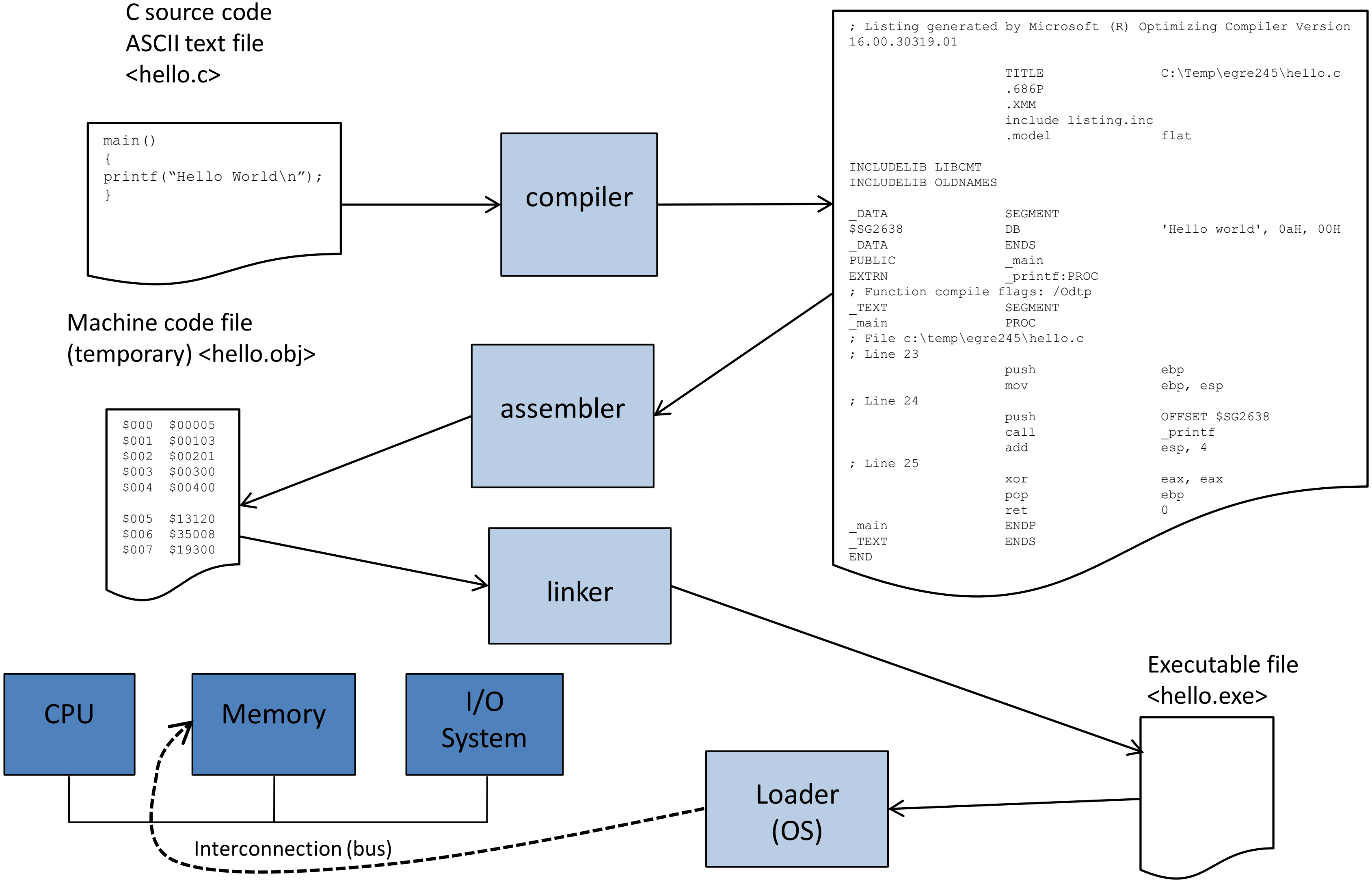
Machine Code

Address	Instruction	Comment
$000	$00005	; LOAD multiplicand, 05
$001	$00103	; LOAD multiplier, 03
$002	$00201	; LOAD bit_mask, 01
$003	$00300	; LOAD result_msb, 00
$004	$00400	; LOAD result_lsb, 00
$005	$13120	; TEST multiplier, bit_mask
$006	$35008	; JUMP Z, no_add
$007	$19300	; ADD result_msb, multiplicand
$008	$20308	; SRA result_msb
$009	$20408	; SRA result_lsb
$00A	$20206	; SL0 bit_mask
$00B	$35405	; JUMP NZ, mult_loop
$00C	$3400C	; JUMP _end_main

Why use the C Language?

- C is a high-level language designed to produce efficient, fast, executable code
- C is one of the few languages that can run on (and for which compilers exist) virtually any size computer – from supercomputers to tiny 8-bit microcontrollers
- The C language allows the programmer to explicitly manage the creation and deletion of data objects and explicitly address specific memory locations
  - This is a requirement for developing programs for hardware-based embedded systems (i.e., microcontroller-based systems)
  - This is not supported in garbage collection-based languages like Java and C#
- Learning (or teaching yourself) a “higher-level” language that includes features like objects, graphics manipulation, or garbage collection is easier after learning a more structured language like C, but the converse is not always true

Program Compile



C source code
ASCII text file
<hello.c>

```
main()
{
printf("Hello World\n");
}
```

compiler

Assembly language file (temporary)
<hello.asm>

```
; Listing generated by Microsoft (R) Optimizing Compiler Version
16.00.30319.01

                    TITLE               C:\Temp\egre245\hello.c
                    .686P
                    .XMM
                    include listing.inc
                    .model              flat

INCLUDELIB LIBCMT
INCLUDELIB OLDNAMES

_DATA               SEGMENT
$SG2638             DB                  'Hello world', 0aH, 00H
_DATA               ENDS
PUBLIC              _main
EXTRN               _printf:PROC
; Function compile flags: /Odtp
_TEXT               SEGMENT
_main               PROC
; File c:\temp\egre245\hello.c
; Line 23
                    push                ebp
                    mov                 ebp, esp
; Line 24
                    push                OFFSET $SG2638
                    call                _printf
                    add                 esp, 4
; Line 25
                    xor                 eax, eax
                    pop                 ebp
                    ret                 0
_main               ENDP
_TEXT               ENDS
END
```

assembler

Machine code file
(temporary) <hello.obj>

```
$000   $00005
$001   $00103
$002   $00201
$003   $00300
$004   $00400

$005   $13120
$006   $35008
$007   $19300
```

linker

Executable file
<hello.exe>

Loader
(OS)

CPU

Memory

I/O
System

Interconnection (bus)