Still More
Assembler Programming

COMP375 Computer Architecture and Organization
“There are 10 types of people: those who understand binary, and those who don’t.”
Assembler Assignment

• The first assembler programming assignment has been posted on Blackboard
• You are required to write four short program segments in assembler
• It is easiest to embed the assembler in C++
• Upload your source code files to Blackboard by noon on Wednesday, September 4, 2019
Goals for Today

• Introduce assembler language constructs for:
  – If statements
  – Loops

• Write simple assembler programs
Division

• The 64 bit number in the EDX:EAX pair of registers is divided by the 32 bit value in a memory location or another register
• Quotient is stored in EAX, remainder in EDX
• Since the EDX:EAX registers are always used, you do not have to specify them
• The divide instruction cannot use a constant, only a memory location or register

    idiv     memoryAddr or reg
Shift Arithmetic

• Shifting a number to the left multiplies it by 2 for each bit you shift
• Shifting a number to the right divides it by 2 for each bit you shift
Shift Example

```c
int dog=3, cat = -4;

_asm {
    mov bx, dog
    sal bx, 2
    sar bx, 1
    mov bx, cat
    sar bx, 1
    mov bx, cat
    shr bx, 1
}
```

**BX in binary**

- `mov bx, dog`: 0000000000000011
- `sal bx, 2`: 0000000000001100
- `sar bx, 1`: 0000000000000110
- `mov bx, cat`: 1111111111111100
- `sar bx, 1`: 1111111111111110
- `mov bx, cat`: 1111111111111110
- `shr bx, 1`: 0111111111111110
Intel Status Register

• The status register records the results of executing the instruction
• Performing arithmetic sets the status register
• Some instructions, such as mov, push or jmp, do not change the status flags
• All jump instructions are based on the status register
# Intel Status Register

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Flag Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>ID Flag (ID)</td>
</tr>
<tr>
<td>30</td>
<td>Virtual Interrupt Pending (VIP)</td>
</tr>
<tr>
<td>29</td>
<td>Virtual Interrupt Flag (VIF)</td>
</tr>
<tr>
<td>28</td>
<td>Alignment Check (AC)</td>
</tr>
<tr>
<td>27</td>
<td>Virtual-8086 Mode (VM)</td>
</tr>
<tr>
<td>26</td>
<td>Resume Flag (RF)</td>
</tr>
<tr>
<td>25</td>
<td>Nested Task (NT)</td>
</tr>
<tr>
<td>24</td>
<td>I/O Privilege Level (IOPL)</td>
</tr>
<tr>
<td>23</td>
<td>Overflow Flag (OF)</td>
</tr>
<tr>
<td>22</td>
<td>Direction Flag (DF)</td>
</tr>
<tr>
<td>21</td>
<td>Interrupt Enable Flag (IF)</td>
</tr>
<tr>
<td>20</td>
<td>Trap Flag (TF)</td>
</tr>
<tr>
<td>19</td>
<td>Sign Flag (SF)</td>
</tr>
<tr>
<td>18</td>
<td>Zero Flag (ZF)</td>
</tr>
<tr>
<td>17</td>
<td>Auxiliary Carry Flag (AF)</td>
</tr>
<tr>
<td>16</td>
<td>Parity Flag (PF)</td>
</tr>
<tr>
<td>15</td>
<td>Carry Flag (CF)</td>
</tr>
</tbody>
</table>
Flag Setting During Execution

```c
int dog=-3, cat=3, bird=5, cow;
_asm { // cow = dog + cat - bird;

    mov eax, dog  
    add eax, cat  
    sub eax, bird  
    mov cow, eax
}
```

<table>
<thead>
<tr>
<th>Zero</th>
<th>Sign</th>
<th>Carry</th>
<th>Overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Compare Instruction

• The `cmp` instruction compares two values and sets the status flags appropriately

  `cmp register, operand`

• where the operand can be a memory location or a register

• The compare instruction subtracts the operand from the register value, but does not save the results
Jump statements

- A **JMP** instruction (sometimes called *branch*) causes the flow of execution to go to a specified location.
- A **JMP** instruction loads the Program Counter with the specified address.
- An unconditional jump always jumps.
- A conditional jump will do nothing if the condition is not met.
- Some architectures have a separate compare instruction.
Jumps Based on Status Flags

JE    Jump if equal    ZF=1
JZ    Jump if zero     ZF=1
JNE   Jump if not equal ZF=0
JNZ   Jump if not zero  ZF=0
JLE   Jump if less or equal ZF=1 or SF≠OF
JL    Jump if less      SF≠OF
JNS   Jump if not sign  SF=0
JS    Jump if sign      SF=1
JGE   Jump if greater or equal SF=OF
JG    Jump if greater   ZF=0 and SF=OF
Labels in Assembler

• You can attach a name to a memory location in assembler. This allows you to use the name instead of numerical address

• Labels start in first column and end with a colon :

```assembly
jmp rabbit
// some other stuff here
rabbit: mov eax, dog
```
Compare and Jump

• A compare instruction sets the flags as if the second operand was subtracted from the first
  \[
  \text{cmp eax, goat}
  \]
  \[
  \text{jge someplace}
  \]
• This will jump if eax ≥ goat
• Imagine the jump comparison is between the register and operand
Jump is all you have

• In assembler there is no
  – if
  – do
  – while
  – switch
  – break

• All changes in execution are done by jump instructions
If statements

• The high level language IF statement is easily implemented by a conditional jump

```c
if (cat >= 13)
    cow = goat;
else
    bull = 47;
```

```assembly
MOV eax, cat
CMP eax, 13
JGE tiger
MOV bull, 47
JMP after
tiger:
    MOV ebx, goat
MOV cow, ebx
after:
```
Forward Jumps

if (cat >= 13) {
  cow = goat;
} else {
  bull = 47;
}

mov eax, cat
 cmp eax, 13
 jge tiger

mov bull, 47
 jmp after

mov ebx, goat

mov cow, ebx

mov ebx, goat

Loops

- Loops are implemented with conditional jumps

```plaintext
while (turtle != 5) {
  // something
}
```

```plaintext
again: mov ebx, turtle
cmp ebx, 5
je endloop
  // something
jmp again
endloop:
```
While Loops

- Loops are implemented with conditional jumps

```java
while (turtle != 5) {
    // something
}
```

```
again: mov    ebx, turtle
    cmp    ebx, 5
    je     endloop
    // something
    jmp    again

endloop:
```
Do While Loops

do {
    // something
} while (bunny < 3);

• Loops involve a jump back

again:
    // something
    mov eax, bunny
    cmp eax, 3
    jl again
In which direction do jumps go?

A. if: forward
   loops: forward
B. if: back
   loops: forward
C. if: forward
   loops: back
D. if: back
   loops: back
Try It

• Write an Intel Assembler program that will set \texttt{big} to 1 if \texttt{num} is greater than 1000 and set \texttt{big} to 0 if it is not

```c
int big, num;
cin >> num;
```

Your assembler goes here

```c
cout << big;
```
Possible Solution

• Write an Intel Assembler program that will set `big` to 1 if `num` is greater than 1000 and set `big` to 0 if it is not.

```
    mov     ebx, num
    cmp     ebx, 1000
    jgt     large
    mov     big, 0
    jmp     around

large:   mov     big, 1
around:  
```
Another Possible Solution

• Write an Intel Assembler program that will set $\text{big}$ to 1 if $\text{num}$ is greater than 1000 and set $\text{big}$ to 0 if it is not

```
mov big, 0
mov ebx, num
cmp ebx, 1000
jle large

large:
```

```
for Loops

• Assembler does not have a for loop. You need to implement it from compares and jumps
• The increment or decrement instruction can be used to implement ++
Example for Loop

// fox = 23;
// for (i = 0; i < 5; i++)
// fox = fox + i;

    mov    ebx, 0 ; ebx holds i
    mov    eax, 23 ; eax hold fox
again:  cmp    ebx, 5 ; end of loop?
        jge     done
        add    eax, ebx ; fox = fox + i
        inc    ebx ; i++
        jmp     again

done:   mov    fox, eax ; save fox
No Operation

• The **NOP** instruction is a one byte instruction that does **nothing**

• Executing a **NOP** does not change any registers or status bits

• NOP instructions can be used to patch machine language, to prevent hazards or to occupy a branch delay slot
While loop example

```cpp
int monkey, chimp;
cin >> monkey >> chimp;
// while (monkey < chimp) {
//     monkey = monkey - chimp;
// }
    mov ebx, monkey
again: cmp ebx, chimp
jge done  // go to done if monkey not < chimp
    sub ebx, chimp
    jmp again
done:    mov monkey, ebx
```
Common Assembler Errors

• Programs generally flow from top to bottom
• If you jump to an earlier (higher on the page) location, you are creating a loop
• An `if` statement does not create a loop
Jump to Next Line (a favorite student error)

```
mov    ebx, dog
cmp    ebx, cow
jg     here
here:  mov    ebx, goat
```

- What happens if dog ≤ cow?
- What happens if dog > cow?
Jump Instruction Implementation

• On several architectures (*not necessarily Intel*) there are only two real jump instructions
• The jump instructions contain a bunch of bits that correspond to the status bits
• The bits in the instruction are logically ANDed with the status bits
• One jump instruction jumps if the result is zero, the other if the result is nonzero
Write the Assembler

• Complete this assembler program to sum the numbers from N down to 1

  mov  ebx, 0  ; ebx is the sum
  mov  ecx, n  ; ecx = n
Possible Solution

• Complete this assembler program to sum the numbers from N down to 1

```
    mov  ebx, 0     ; ebx is the sum
    mov  ecx, n     ; ecx = n

  wloop: cmp  ecx, 0     ; while N != 0
       jz   done       ; end of loop if zero
       add  ebx, ecx    ; sum += n
       dec  ecx         ; n--
       jmp  wloop       ; repeat

  done:   mov  sum, ebx   ; save sum
```
Another Possible Solution

• Complete this assembler program to sum the numbers from N down to 1

```
    mov  ebx, n ; ebx = n
    mov  ecx, n ; ecx = n
    inc  ecx ; ecx = n + 1
    imul ebx, ecx ; ebx = n * (n+1)
    sar  ebx, 1 ; ebx = (n * (n+1)) / 2
    mov  sum, ebx ; save result
```
Addresses

• Assembler programs often have to manipulate addresses
• A pointer in C++ represents an address in assembler
• You may need to use addresses to follow links in a data structure or to get an element from an array
Intel Assembler Addresses

• You can load a register with the address of a memory location by using the Load Effective Address, `lea`, instruction

  \[
  \text{lea eax, dog} \quad ; \text{eax = address of dog}
  \]

• If the memory location is based on indexing, the `lea` instruction will compute the correct address

  \[
  \text{lea eax, dog[esi]} \quad ; \text{effective address}
  \]
Indexed Addressing

• When you write a register in [brackets], this means to use the address in the register

\[
\text{mov ebx, [eax]}
\]

• ebx will get the value in memory whose address is in eax
Index and Addresses

• If you put a register in [brackets] after a memory address, this means the hardware will add the value in the register to the address to get the effective address to use:

```
mov ebx, hawk[eax]
```

• ebx will get the value in memory whose address is the sum of the address hawk plus the value in the eax register.
Indexing Arrays

• An array is a sequential collection of data values at consecutive addresses
• The first element of an array (index 0) is at the start address of the array
• The second element’s address is the start address of the array plus the size of each element
Indexing

• To specify that the address of the data is in a register in Intel assembler, you put the register in the operand field in [brackets]

```c
// char cat[47], goat;
// goat = cat[10];
lea ebx, cat ; ebx = addr of cat
add ebx, 10 ; add 10 to address
mov al, [ebx] ; al = cat[10]
mov goat, al ; save in goat
```
Program to Sum 5 Numbers

```c
int sum = 0;
    /* Sum in C++ */
int i;
for (i = 0; i < 5; i++) {
    sum = sum + arrayA[i];
}
```
/* Sum in Assembler */

```asm

lea edx, arrayA ; esi = start addr of arrayA
mov eax, 0 ; eax = 0, initialize sum
mov ebx, 0 ; ebx = 0, loop counter

forloop:
    add eax, [edx] ; add next value of array to eax
    add edx, 4 ; increment edx to next element
    inc ebx ; increment loop counter
    cmp ebx, 5 ; end of loop?
    jl forloop ; repeat if not 5
    mov sum, eax ; move result to sum

}```
/* Sum in Assembler */

__asm{
    mov edx, 0 ; edx is index into array
    mov eax, 0 ; eax = 0, initialize sum
    mov ebx, 0 ; ebx = 0, loop counter

forloop:
    add eax, arrayA[edx] ; add next value of array
    add edx, 4 ; increment edx to next element
    inc ebx ; increment loop counter
    cmp ebx, 5 ; end of loop?
    jl forloop ; repeat if not 5
    mov sum, eax ; move result to sum
}

Accessing Fields in an Object

- Consider an object with several data fields.
- Assume register EDX contains the address of the object.
- You can load the count value of the object into EAX by
  \[ \text{mov eax, 8[edx]} \]
Summing the count of all Widgets in a linked list

```
mov edx, prt2first ; get pointer to list
mov eax,0 ; initialize sum to zero
next:
  cmp edx, 0 ; is pointer null
  je done ; go to done if null
  add eax, 8[edx] ; add count of Widget
  mov edx, C[edx] ; get pointer to next Widget
  jmp next ; loop
done: mov sum, eax ; save sum
```
Assembler Assignment

• The first assembler programming assignment has been posted on Blackboard
• You are required to write four short program segments in assembler
• It is easiest to embed the assembler in C++
• Upload your source code files to Blackboard by noon on Wednesday, September 4, 2019
Labor Day

There will be no classes on Monday, September 2