Lecture Overview

- How to write in x86 assembly:
  - do while loops, while loops, for loops, switch statements
  - Some more examples like factorial, string length, finding max in an integer array etc

“do while” example

```
result = 1;
while(n>1):
  result*=n;
  n = n-1;
```

Argument: $n$ at `%ebp+8` and `result` in `%eax`

```
1 movl 8(%ebp), %edx
2 movl $1, %eax
3 .L2:
4 imull %edx, %eax
5 subl $1, %edx
6 cmpl $1, %edx
7 jg .L2
8 loop
9 ret
```

Class Announcements

1. Grades for Programming Assignment 0 have been released in learn@UW.
2. If you have questions about your grading please contact Lokesh or Urmish.
### "while" example

```c
result = 1;
while(n>1){
  result*=n;
  n = n-1;
};
```

**Argument:** \( n \) at %ebp+8

**Registers:**
- \( n \) in %edx
- result in %eax

1. movl $\%ebp, %edx  
   get \( n \)
2. movl $1, %eax  
   result = 1
3. cmp $1, %edx  
   compare \( n \):1
4. jle .L7  
   If \( \leq \), goto done
5. .L10:
6. imull %edx, %eax  
   \( \text{result}\) *= \( n \)
7. subl $1, %edx  
   decrement \( n \)
8. cmp $1, %edx  
   compare \( n \):1
9. jg .L10  
   If \( > \), goto loop
10. .L7:  
    done:
11. ret
```

### FOR LOOP EXAMPLE

\[
\sum i = \frac{n(n+1)}{2}
\]

```c
sum = 0;
for (i = 1; i < N; i++) {
    sum = sum + i;
}
```

### Karen’s implementation:

```assembly
movl N, %ecx
movl $0, %eax  
sum in eax
movl $1, %edx  
i in edx
.L5:    cmp %edx, %ecx  
.j1 .L6  
jump when \( i < 0 \)
addl %edx, %eax  
.incl %edx
jmp .L5
.L6:
```

### gcc’s implementation (mostly):

```assembly
movl N, %ecx
movl $0, %eax  
sum in eax
movl $1, %edx  
i in edx
jmp .L2
.L3:    addl %edx, %eax  
sum = sum + i
.incl %edx
.L2:    cmp %ecx, %edx  
.jle .L3  
jump when \( i \geq 0 \)
```

Jump when \( \text{less than or equal to} \ 0 \)
About Switch Statement and Jump Tables

1. Switch statements offer multi-way branching capability and are implemented using jump tables which are supported by GCC as an extension to C.

2. Jump table is an array where the i\textsuperscript{th} entry is the address of the code segment that should execute when the switch index equals i.

3. Advantage of Jump tables when compared to long sequence of compares and jumps: Time taken to perform the switch is independent of the number of cases and the sparsity of the case values.

4. Jump tables used only when there are a number of cases (4 or more) and they span a small range of values.

Conditional Move Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Synonym</th>
<th>Move condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmov E, R</td>
<td>cmov E</td>
<td>zF</td>
<td>Equal / zero</td>
</tr>
<tr>
<td>cmov E, R</td>
<td>cmov E</td>
<td>¬zF</td>
<td>Not equal / not zero</td>
</tr>
<tr>
<td>cmov E, R</td>
<td>cmov E</td>
<td>¬SF</td>
<td>Negative</td>
</tr>
<tr>
<td>cmov E, R</td>
<td>cmov E</td>
<td>¬SF</td>
<td>Negative</td>
</tr>
<tr>
<td>cmov E, R</td>
<td>cmov E</td>
<td>¬SF</td>
<td>Negative</td>
</tr>
<tr>
<td>cmov E, R</td>
<td>cmov E</td>
<td>¬SF</td>
<td>Negative</td>
</tr>
<tr>
<td>cmov E, R</td>
<td>cmov E</td>
<td>¬SF</td>
<td>Negative</td>
</tr>
<tr>
<td>cmov E, R</td>
<td>cmov E</td>
<td>¬SF</td>
<td>Negative</td>
</tr>
</tbody>
</table>

Figure 3.17: The conditional move instructions. These instructions copy the source value \( S \) to its destination \( D \) when the move condition holds. Some instructions have "synonyms," alternate names for the same machine instruction.

Example x86 programs

- Factorial
- Find max in integer array
- String length
- Count the bits set in an integer - popcount