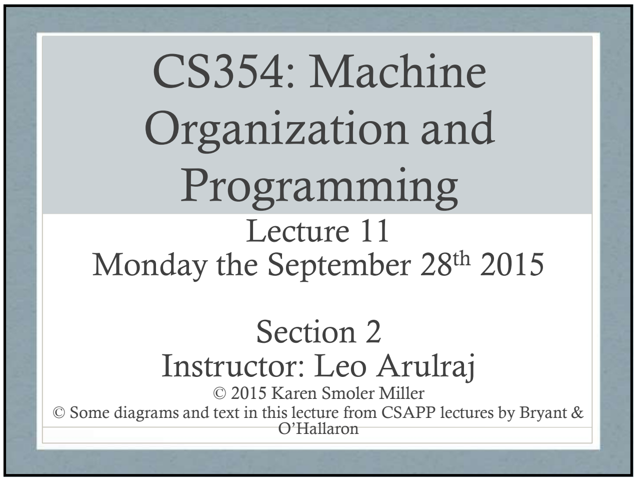
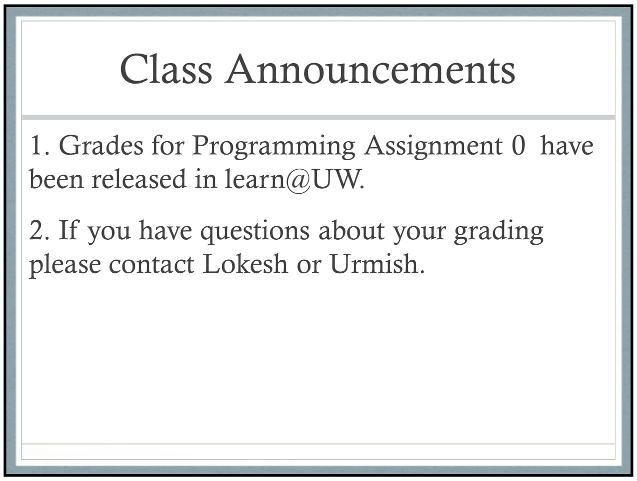


# CS354: Machine Organization and Programming

Lecture 11
Monday the September 28th 2015

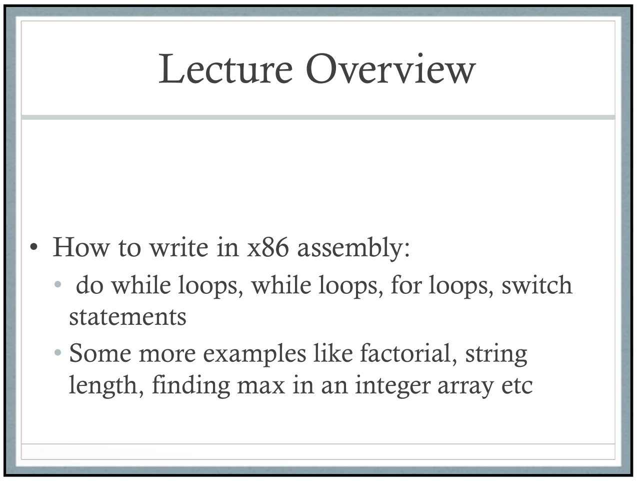
Section 2
Instructor: Leo Arulraj



## Class Announcements

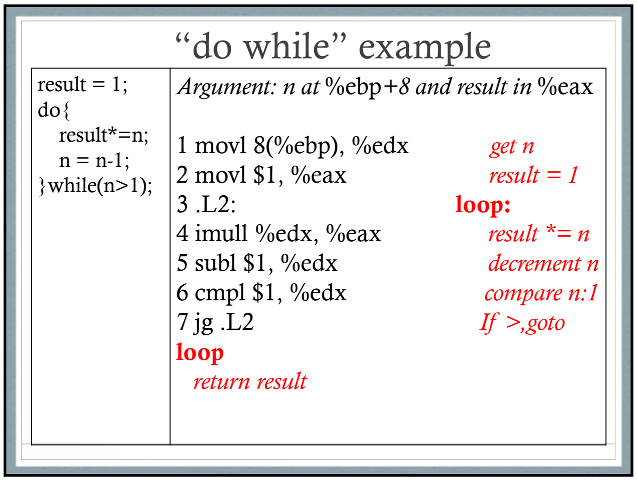
1. Grades for Programming Assignment 0 have been released in learn@UW.

2. If you have questions about your grading please contact Lokesh or Urmish.



## 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

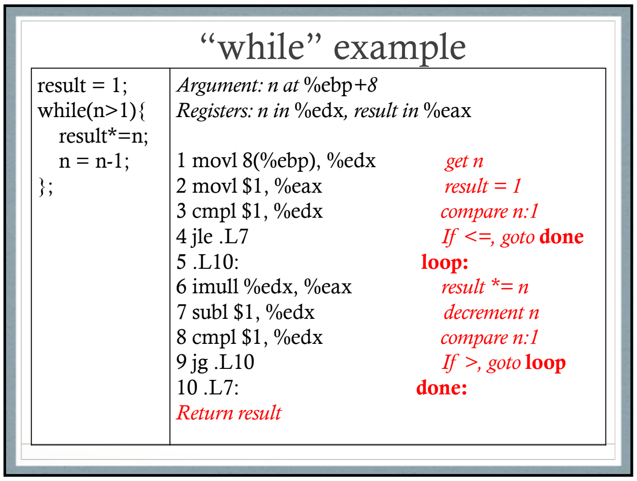


## "do while" example

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

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

```
movl 8(%ebp), %edx        get n
movl $1, %eax             result = 1
.L2:                  loop:
imull %edx, %eax          result *= n
subl $1, %edx             decrement n
cmpl $1, %edx             compare n:1
jg .L2                    If >,goto
loop
  return result
```

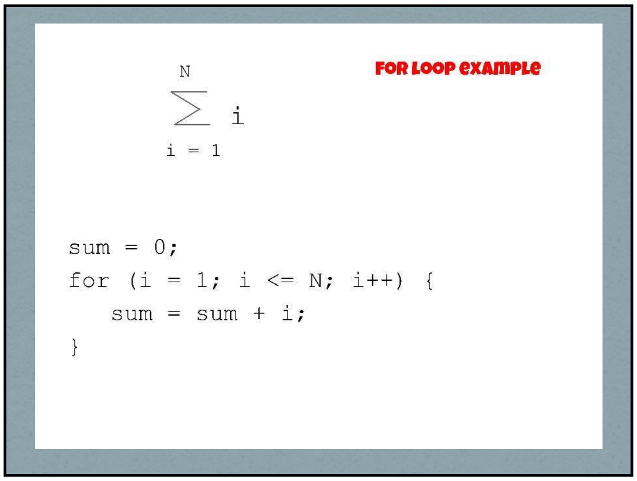


## “while” example

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

*Argument: n at* %ebp+*8*
*Registers: n in* %edx, *result in* %eax

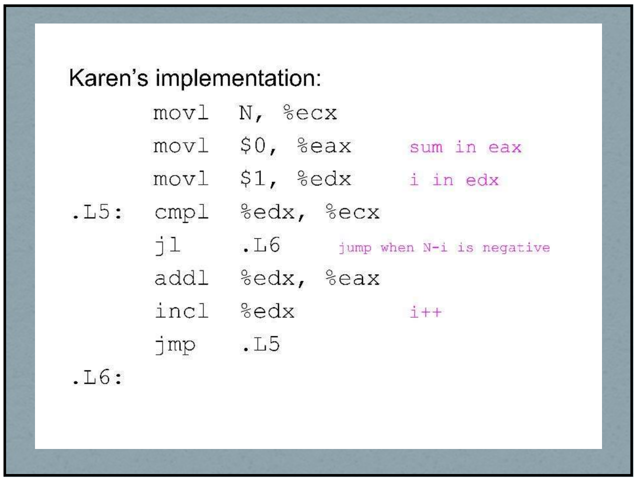
```
movl 8(%ebp), %edx      get n
movl $1, %eax           result = 1
cmpl $1, %edx           compare n:1
jle .L7                 If <=, goto done
.L10:                   loop:
imull %edx, %eax        result *= n
subl $1, %edx           decrement n
cmpl $1, %edx           compare n:1
jg .L10                 If >, goto loop
.L7:                    done:
Return result
```



FOR LOOP EXAMPLE

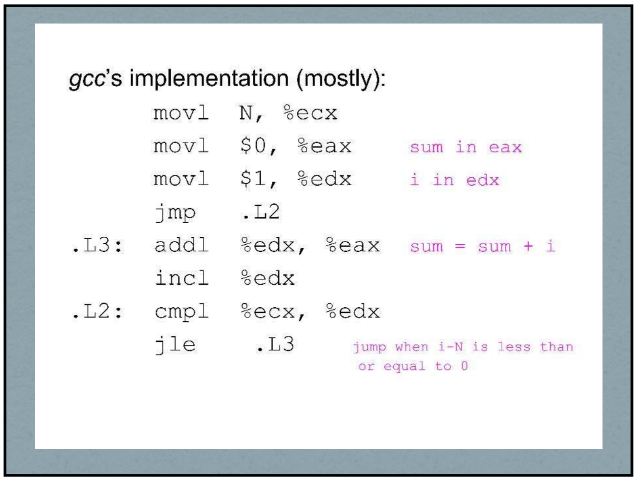
$$\sum_{i = 1}^{N} i$$

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



Karen’s implementation:

```
       movl   N, %ecx
       movl   $0, %eax     sum in eax
       movl   $1, %edx     i in edx
.L5:   cmpl   %edx, %ecx
       jl     .L6        jump when N-i is negative
       addl   %edx, %eax
       incl   %edx         i++
       jmp    .L5
.L6:
```



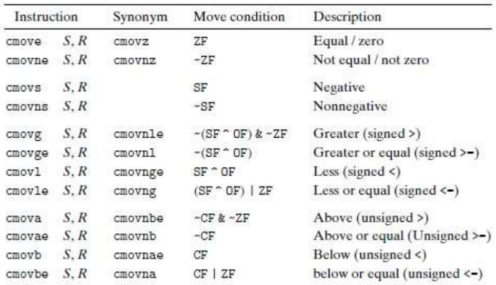
*gcc*’s implementation (mostly):

```
       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:   cmpl   %ecx, %edx
       jle     .L3       jump when i-N is 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$^{th}$ 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

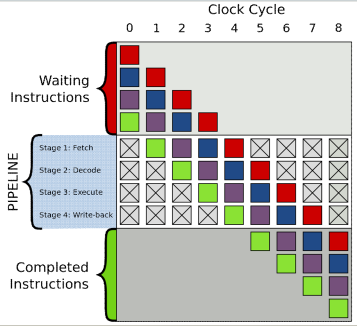
## Conditional Move Instructions



| Instruction | | Synonym | Move condition | Description |
|---|---|---|---|---|
| cmove | *S*, *R* | cmovz | ZF | Equal / zero |
| cmovne | *S*, *R* | cmovnz | ~ZF | Not equal / not zero |
| cmovs | *S*, *R* | | SF | Negative |
| cmovns | *S*, *R* | | ~SF | Nonnegative |
| cmovg | *S*, *R* | cmovnle | ~(SF ^ OF) & ~ZF | Greater (signed >) |
| cmovge | *S*, *R* | cmovnl | ~(SF ^ OF) | Greater or equal (signed >-) |
| cmovl | *S*, *R* | cmovnge | SF ^ OF | Less (signed <) |
| cmovle | *S*, *R* | cmovng | (SF ^ OF) \| ZF | Less or equal (signed <-) |
| cmova | *S*, *R* | cmovnbe | ~CF & ~ZF | Above (unsigned >) |
| cmovae | *S*, *R* | cmovnb | ~CF | Above or equal (Unsigned >-) |
| cmovb | *S*, *R* | cmovnae | CF | Below (unsigned <) |
| cmovbe | *S*, *R* | cmovna | CF \| ZF | below or equal (unsigned <-) |

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

## Pipelining and Conditional Move (Refer 3.6.6 in CSAPP textbook)



## Example x86 programs

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