# 15-213 Recitation 2: 09/16/02

## Outline

- Assembly walkthrough
- Using GDB
- Play with examples
  - C to assembly
  - Assembly to C


**Annie Luo**

**e-mail:**

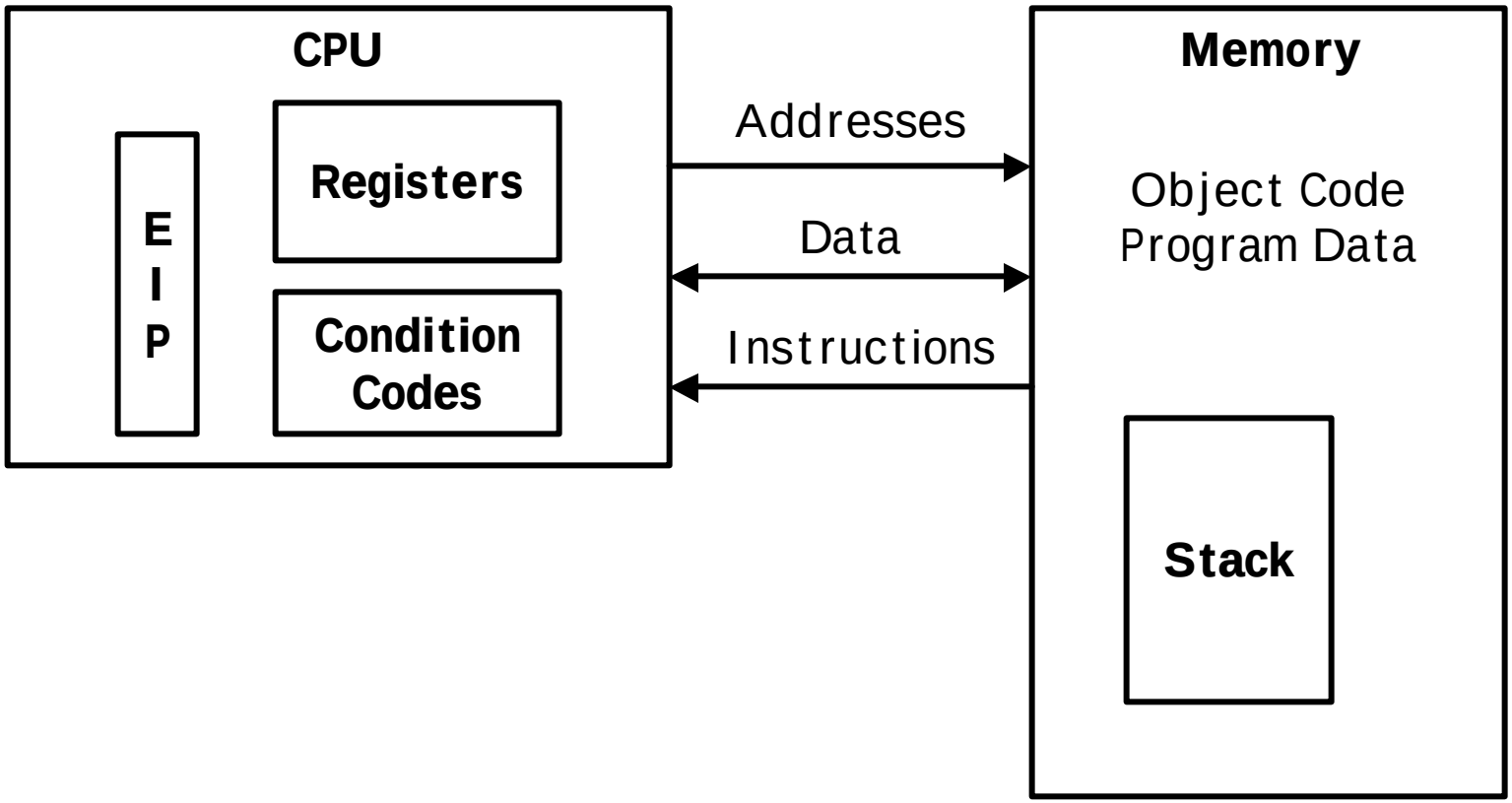
luluo@cs.cmu.edu


**Office Hours:**

Thursday 6:00 – 7:00pm

Wean 8402

# Machine Model



# Special Registers

- %eax Return Value
- %eip Instruction Pointer
- %ebp Base (Stack Frame) Pointer
- %esp Stack Pointer

# Assembly Programming: Structure

## Function Setup

- Save old base pointer (`pushl %ebp`)
- Set up own base pointer (`movl %esp, %ebp`)
  - Note that this saves the old stack pointer

## Function Body

- Operations on data, loops, function calls

# Assembly Programming: Structure

## Function Cleanup

- Return value placed in %eax
  - What about returning larger values? (structs, doubles, etc.)
- Restore Caller's Stack Pointer (`movl %ebp,%esp`)
- Restore Old Base Pointer (`popl %ebp`)
- Return
  - Where does it return to?

# Assembly Programming: Simple Addressing Modes

## Examples

- (R) Mem[R]
- $10(R) Mem[R + 10]
- $0x10(R) Mem[R + 16]

# Assembly Programming: Indexed Addressing Modes

## Generic Form

D(Rb, Ri, S)  Mem[Reg[Rb]+S*Reg[Ri]+ D]

## Examples

- (Rb,Ri)  Mem[Reg[Rb]+Reg[Ri]]
- D(Rb,Ri)  Mem[Reg[Rb]+Reg[Ri]+D]
- (Rb,Ri,S)  Mem[Reg[Rb]+S*Reg[Ri]]

# Using GDB

- GNU dubugger
- Best friend for Lab 2
- Usage

  **`gdb <name_of_file>`**

- Get help:
  - Refcard
  - In gdb type “help”
  - Search on web!

# Useful GDB Commands

For Lab 2

- disassemble – disassembles machine code into assembly
- x – examine memory contents

In General

- run – run the program
- break – set breakpoints
- step, next – step through code
- print – print values of variables and memory

# Example 1: Arithmetic

Convert to Assembly

```
int func1(int a, int b)
{
  int x, y;

  x = a + b;
  y = 2*x - b;
  return x*y;
}
```

# Example 2: If Statement

Convert to Assembly

```
int func2(int a, int b)
{
  if (a > b)
    return a;
  else return b;
}
```

# Example 3: switch statements

```
int func3(int a, int b)
{
  int r;

  switch(a) {
  case 0: r = a; break;
  case 1: r = b; break;
  case 2: r = a+b; break;
  case 3: r = a-b; break;
  case 4: r = a*b; break;
  default: r = strlen(MY_STRING); break;
  }
  return r;
}
```

# Audience Participation Time ☺

Convert to C

```
get_sum:
        pushl %ebp
        movl %esp,%ebp
        pushl %ebx

        movl 8(%ebp),%ebx
        movl 12(%ebp),%ecx
        xorl %eax,%eax
        movl %eax,%edx
        cmpl %ecx,%eax
        jge .L4
.L6:    addl (%ebx,%edx,4),%eax
        incl %edx
        cmpl %ecx,%edx
        jl .L6

.L4:    popl %ebx
        movl %ebp,%esp
        popl %ebp
        ret
```

# Audience Participation Time ☺

Convert to C

```
get_sum:
        pushl %ebp
        movl %esp,%ebp
        pushl %ebx

        movl 8(%ebp),%ebx          # ebx = 1st arg
        movl 12(%ebp),%ecx         # ecx = 2nd arg
        xorl %eax,%eax             # eax = 0
        movl %eax,%edx             # edx = 0
        cmpl %ecx,%eax             #
        jge .L4                    # if (ecx <= 0) goto L4
.L6:    addl (%ebx,%edx,4),%eax    # eax += Mem[ebx+edx*4]
        incl %edx                  # edx ++
        cmpl %ecx,%edx             #
        jl .L6                     # if (ecx < edx) goto L6

.L4:    popl %ebx
        movl %ebp,%esp
        popl %ebp
        ret
```

# Audience Participation: Answer

```
int get_sum(int * array, int size)
{
    int sum = 0;
    int i=0;
    for (i=0; i<size; i++)
        sum += array[i];
    return sum;
}
```

# Example 5

A mystery function

```
int mystery(int x, int y)
```

is compiled into the assembly on the next page.

What is the function?

# Example 5

```
A0:  mystery:
A1:     movl   8(%ebp), %edx
A2:     movl   $0x0, %eax
A3:     cmpl   $0x0, 12(%ebp)
A4:     je     .L11
A5: .L8
A6:     cmpl   $0x0, 12(%ebp)
A7:     jle    .L9
A8:     add    %edx, %eax
A9:     decl   12(%ebp)
A10:    jmp .L10
A11:.L9
A12:    sub    %edx, %eax
A13:    incl   12(%ebp)
A14:.L10
A15:    compl $0x0, 12(%ebp)
A16:    jne .L8
A17:.L11
A18:    mov %ebp, %esp
A19:    pop %ebp
A20:    ret
```

```
# Get x
# Set result = 0
# Compare y:0
# If(y == 0) goto Done
# Loop:
# Compare y:0
# If(y <= 0) goto Negative
# result += x
# y- -
# goto Check
# Negative:
# result -= x
# y++
# Check:
# Compare y:0
# If(y != 0) goto Loop
# Done:
#
# Cleanup
#
```

# Example 5

So what is the function computing?

```
/*  x times y, where x and y may be pos. or neg. */

int multiply(int x, int y)
{
   int result = 0;
   while( y != 0 ) {
      if (y > 0) {
         result += x;
         y--;
      }
      else {
         result -= x;
         y++;
      }
   }
   return result;
}
```