Homework / Exam

- Reading
  - PAL, pp 216-227
- Homework
  - mp2 due before class number 12
- Exam #1
  - Class 13 (three sessions from today)
  - Open book / Open notes
  - Practice exam posted on web site now

Using C Structs in Assembly Code

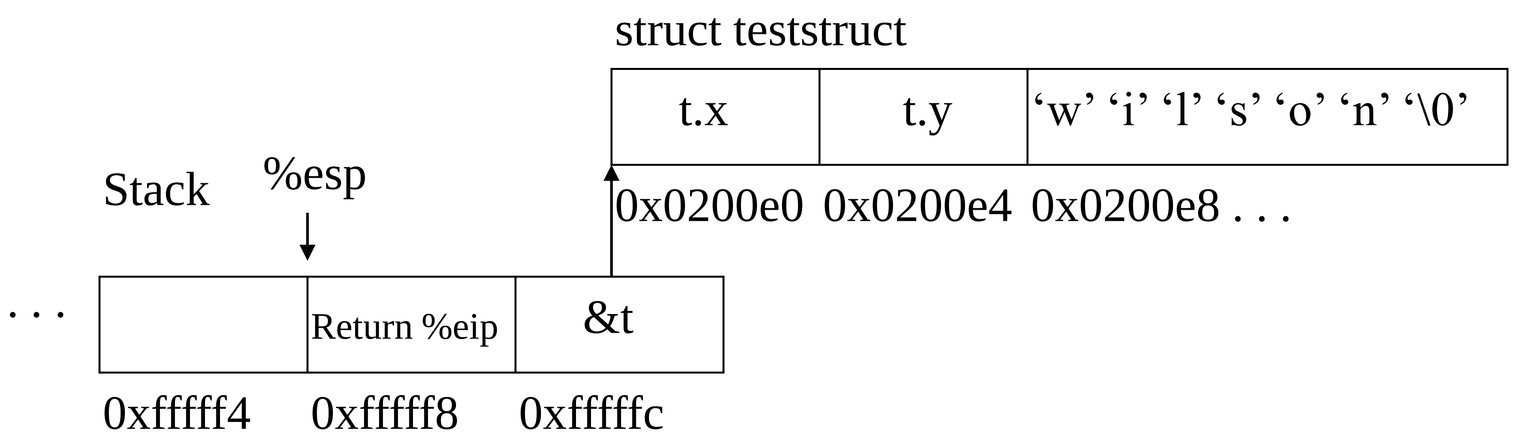
- How do we access a C structure such as:

```
#define NAMELEN 20
struct teststruct {
        int x,
        int y;
        char name[NAMELEN];
}t;
t.x = 2;
t.y = 5;
strncpy(t.name, “wilson”, NAMELEN);
trystruct(&t);    /* pass to asm via pointer*/
```

Using C Structs in Assembly Code

- Assembly code would look like:

```
movl  4(%esp),%edx # ptr to t
movl  (%edx),%eax  # x itself
movl  4(%edx),%ebx # y itself
movb  8(%edx),%cl  # 1st string char
```



Using C Structs in Assembly Code

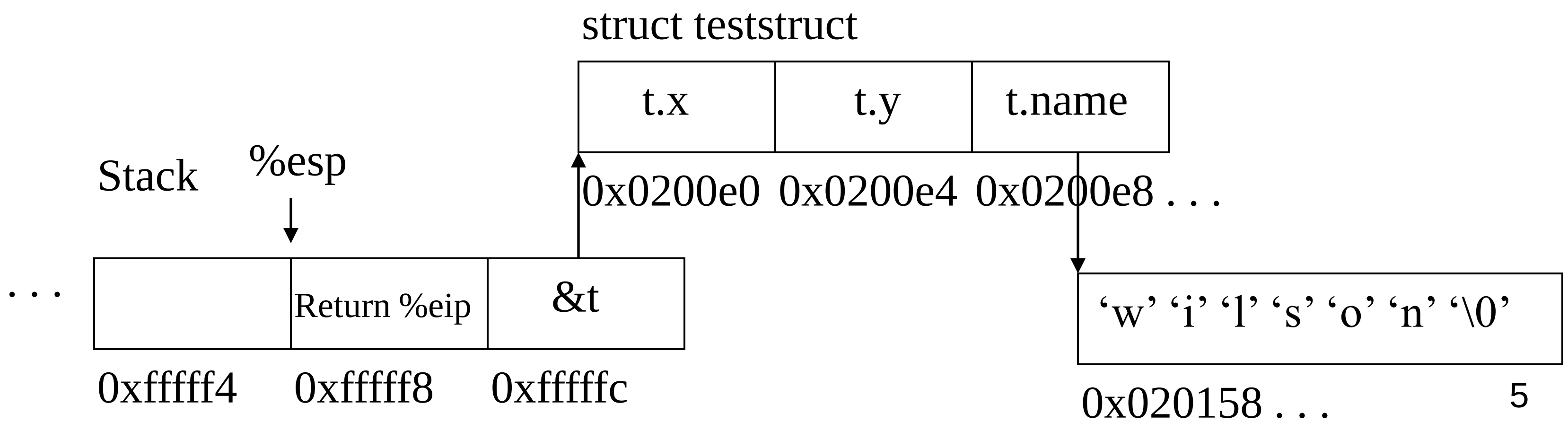
- However, we would normally have a pointer to string:

```
#define NAMELEN 20
char array [NAMELEN];
struct teststruct {
        int x,
        int y;
        char *name;
}t;
t.x = 2;
t.y = 5;
t.name = array;
strncpy(array, “wilson”, NAMELEN);
trystruct(&t);    /* pass to asm via pointer*/
```

Using C Structs in Assembly Code

- Assembly code would look like:

```
movl   4(%esp),%edx    # ptr to t
movl   (%edx),%eax     # x itself
movl   4(%edx),%ebx    # y itself
movl   8(%edx),%edx    # ptr to string
movb   (%edx),%cl      # first string char
```

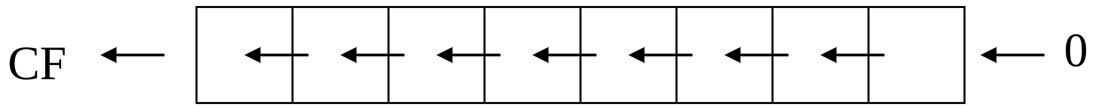


Introduction to Shift Instructions

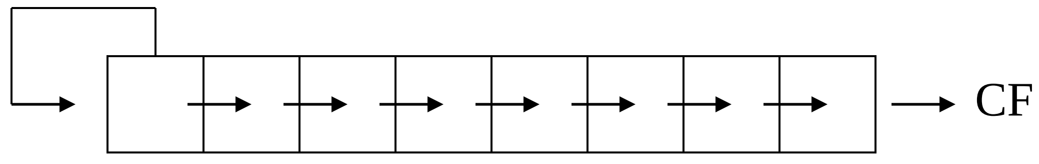
- We can shift the bits in a byte, word, or long word by a variable number of positions
- These are the machine level instructions used to implement the C language operators << and >>
  - SAL / SHL are the left shift instructions for signed or unsigned data (arithmetic or logical left shift)
  - SAR is the right shift instruction for signed data (arithmetic right shift)
  - SHR is the right shift instruction for unsigned data (logical right shift)

Introduction to Shift Instructions

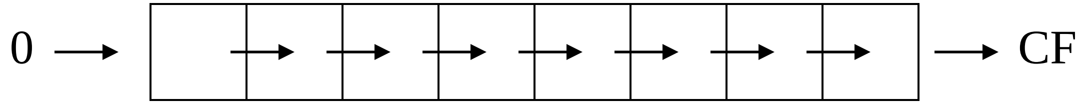
- The SAL / SHL Instruction (Signed / Unsigned)



- The SAR Instruction (Signed)



- The SHR Instruction (Unsigned)



Introduction to Shift Instructions

- The target of the shifting can be a register or memory location (byte, word, or long word)
- The count for the number of bits to shift can be specified with immediate data (constant) or the %cl register (variable)
- Examples:

```
sall $4, %eax     # logical left shift of %eax by 4 bits
sarb %cl, label   # arithmetic right shift of memory byte
                  # by a variable value stored in the %cl
```

Introduction to Shift Instructions

- Multiplication by 2^N can be done via left shift

```
sall $4, %eax     # %eax times 2^4
```

- Can combine left shifts and addition

- Division by 2^N can be done via right shift

```
sarb %cl, label  # memory byte / 2^%cl
```

- Can combine right shifts and subtraction

Introduction to Multiply and Divide

- Unsigned Multiply and Divide
  - mul
  - div
- Signed Multiply and Divide
  - imul
  - idiv
- We won’t do much with these because of the complexity involved - especially for divide

Introduction to Multiply and Divide

- Multiply always operates with %al, %ax, or %eax
- Result needs more bits than either operand
- Syntax:

  mulb %bl

  %ax ← %al * %bl

  mulw %bx

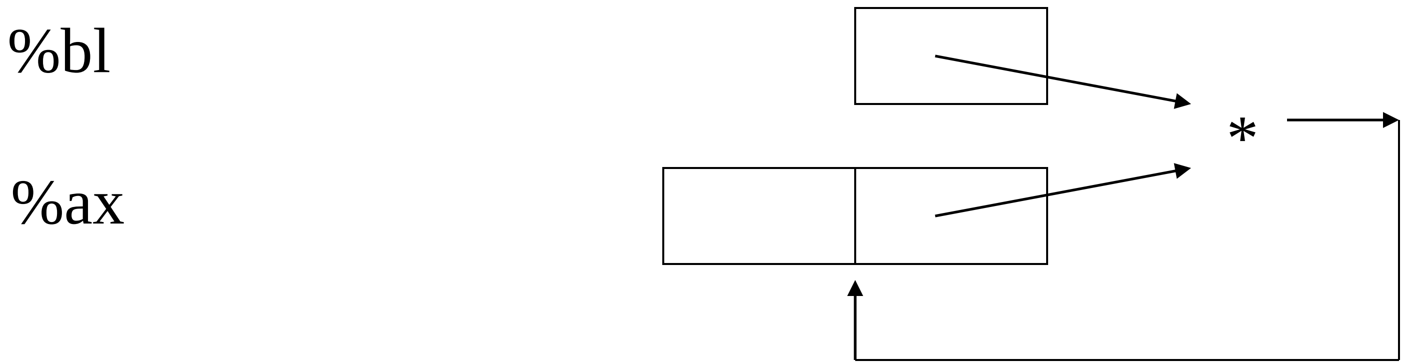
  %dx, %ax ← %ax * %bx

  mull %ebx

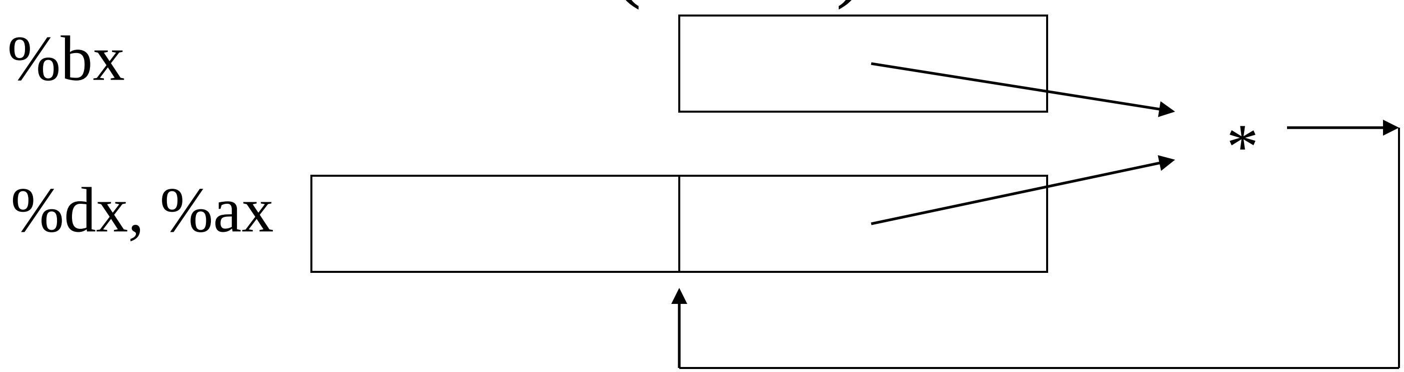
  %edx, %eax ← %eax * %ebx

Introduction to Multiply and Divide

- Register Pictures (Byte)



(Word)



Example – For/While Loop and mul

- C code for n = 5! (done as a for loop)

```
unsigned int i, n;
n = 1;
for (i = 1; i <= 5; i++)
    n *= i;
```

- C code for n = 5! (done as a while loop)

```
unsigned int i, n;
n = i = 1;
while (i <= 5)
  n *= i++;
```

Example – For/While Loop and mul

- Assembly code for n = 5! (byte * byte = word)

```
        movb  $1, %bl      # i = 1
        movb  %bl, %al     # n = i = 1
loop:   cmpb  $5, %bl      # while (%bl <= 5)
        ja    exit         # %bl > 5 now
        mulb %bl           # %ax = %al * %bl
        incb %bl           # incr %bl
        jmp  loop          # and loop
exit:                      # 5! in %ax
```

- Note: No difference between for and while in assy

Example – For/While Loop and mul

- Assembly code for n = 5! (word * word = long)

```
          movw  $1, %bx        # i = 1
          movw  %bx, %ax       # n = i = 1
loop:     cmpw  $5, %bx        # while (%bx <= 5)
          ja    exit           # %bx > 5 now
          mulw %bx             # %ax = %ax * %bx
                               # %dx = 0 now
          incw %bx             # incr %bx
          jmp  loop            # and loop
exit:                          # 5! in %eax
```

Recursive Factorial

- Main program to call recursive factorial subr

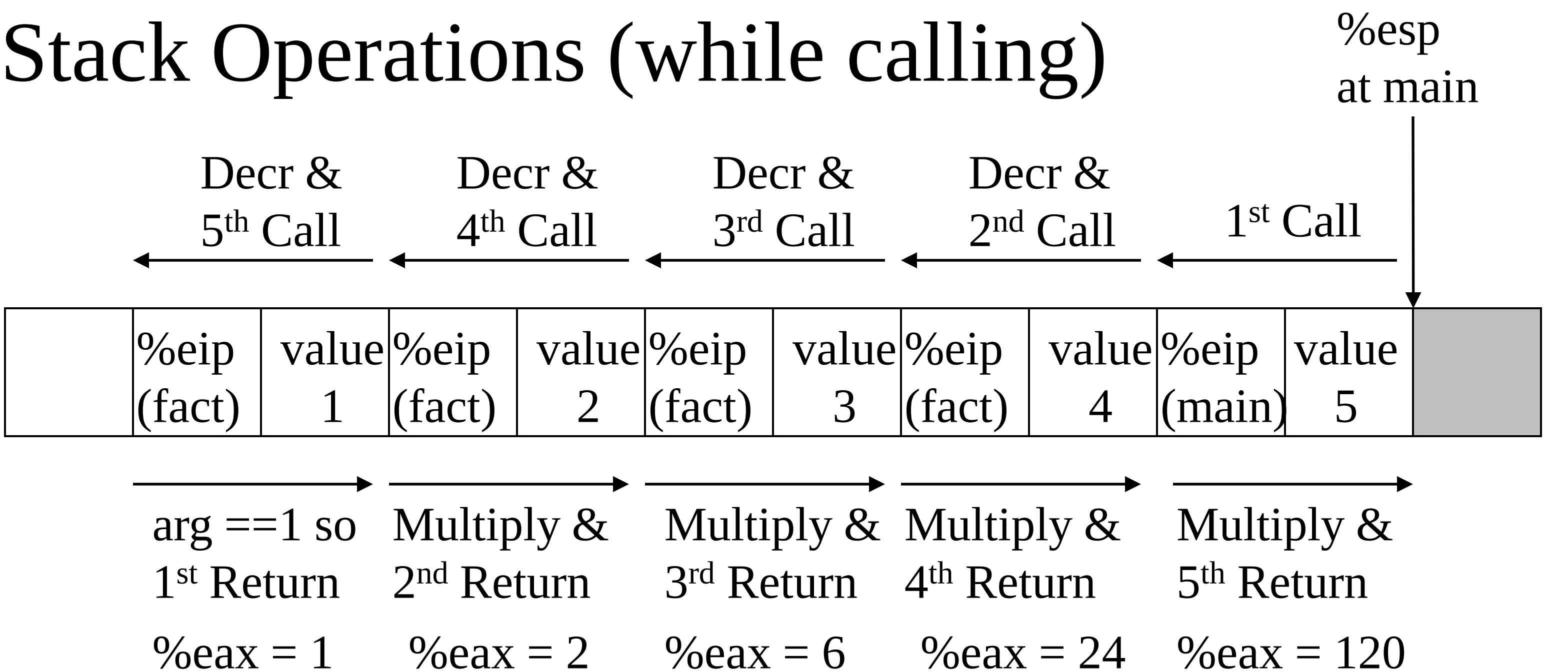
```
.text
pushl   $5
call    factorial
addl    $4, %esp
ret
```

Recursive Factorial

```
factorial:        # works up to 16 bit results
        movl      4(%esp), %eax
        cmpl      $1, %eax
        jna       return
        decl      %eax
        pushl     %eax
        call      factorial
        addl      $4, %esp
        movw      4(%esp), %bx
        mulw      %bx      # 16 lsbs go to %ax
return:                    # ignore msbs in %dx
        ret
        .end
```

Recursive Factorial

- Stack Operations (while calling)



- Stack Operations (while returning)