Intel IA-32 Architecture

CS520
Dept. of Computer Science
Univ. of New Hampshire

Main Lecture Goal

understand how function calls are supported on the Intel IA-32

- recursion
- return addresses
- return values
- parameters

necessary in order to understand how to implement garbage collectors, threads, etc.

Intel IA-32

32-bit addresses

32-bit integer registers

eax, ecx, edx - caller saved

ebx, esi, edi - callee saved

esp - stack pointer

ebp - frame pointer

eip - instruction pointer (PC)

80-bit floating-point registers

internally Intel stores floating-point values in its own non-standard format

values are converted to standard IEEE formats when written to memory

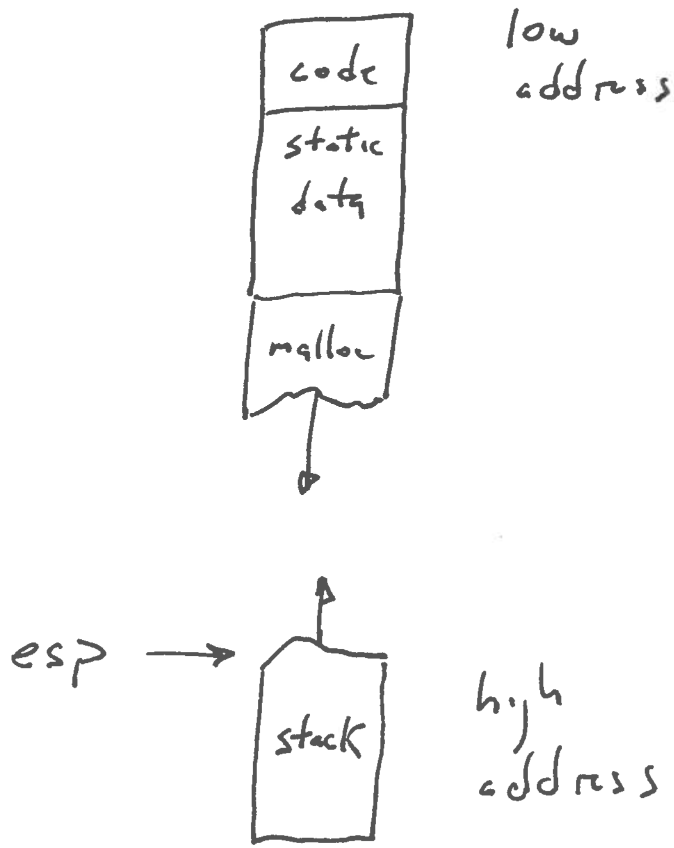
Operand Types

byte	–	8 bits	b
word	–	16 bits	w
long	–	32 bits	l

stack

grows from high address down to
low address

esp points to top of stack



frames

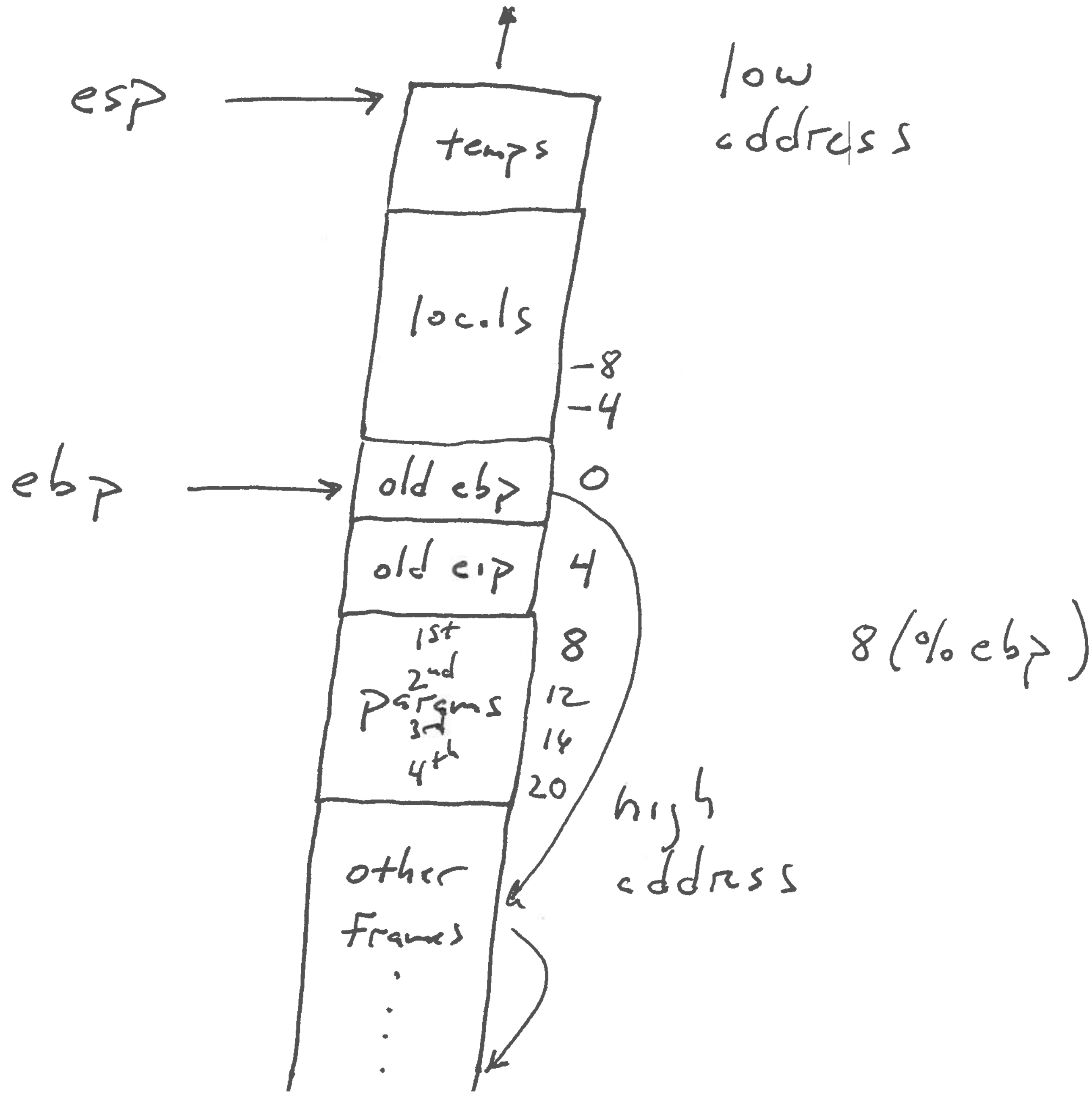
stack contains a series of frames

one for each active function call

each frame contains:

- return address
- saved registers
- local variables
- parameters
- temporaries

ebp points to the top frame on the stack


esp
low address
temps
locals
-8
-4
ebp
old ebp
0
old eip
4
1st
2nd
params
3rd
4th
8
12
14
20
8(%ebp)
high address
other frames
...

return value

integer return values are returned in eax

saving / restoring registers

calling function is responsible for saving and restoring eax, ecx and edx if it needs them upon return — caller saved

→ saved before call
restored after return

called function should save and restore ebx, esi and edi if it uses them — callee saved

→ saved upon entry
restored before return

```
#
# Intel IA-32 (Linux) assembler source for computing the factorial function.
#
# The code is based on MIPS code from pages A-26 and A-27 of Patterson &
# Hennessy.
#
# It computes fact(10).
#
        .text
        .align  4
        .globl  main
main:
        pushl   %ebp                # Save old frame pointer
        movl    %esp,%ebp           # Establish new frame pointer
#
        pushl   $10                 # Put argument (10) on stack
        call    fact                # Call factorial function
        addl    $4,%esp             # Remove argument from stack
#
        pushl   %eax                # Return value from fact pushed to be arg 2
        pushl   $.LC0               # Push format string to be arg 1
        call    printf              # Call the printf function
        addl    $8,%esp             # Remove arguments from stack
#
        popl    %ebp                # Restore frame pointer
        ret                         # Return to caller

#
        .data
.LC0:
        .string "The factorial of 10 is %d\n"
#
#
# The factorial function itself
#
# ie fact(n)
#
        .text
        .align  4
        .globl  fact                # .globl also allows gdb to see label
fact:
        pushl   %ebp                # Save old frame pointer
        movl    %esp,%ebp           # Establish new frame pointer
#
        cmpl    $0, 8(%ebp)         # Test n against 0
        jg      .L2                 # Branch if n > 0
        movl    $1,%eax             # Return 1
        jmp     .L1                 # Jump to code to return
#
.L2:
        movl    8(%ebp),%eax        # Get n
        subl    $1,%eax             # Compute (n - 1)
        pushl   %eax                # Push it to pass as argument
        call    fact                # Recursive call
        addl    $4,%esp             # Remove argument
#
        imull    8(%ebp),%eax       # Compute n * fact(n - 1)
#
.L1:                                # Result is in %eax
        popl    %ebp                # Restore frame pointer
        ret                         # Return to caller
```

~cs520/public/fact.s

gcc fact.s -o fact

```
#
# Intel IA-32 (Linux) assembler source for computing the factorial function.
#
# The code is based on MIPS code from pages A-26 and A-27 of Patterson &
# Hennessy.
#
# It computes fact(10).
#
        .text
        .align  4
        .globl  main
main:
        pushl   %ebp                    # Save old frame pointer
        movl    %esp,%ebp               # Establish new frame pointer
#
        pushl   $10                     # Put argument (10) on stack
        call    fact                    # Call factorial function
        addl    $4,%esp                 # Remove argument from stack
#
        pushl   %eax                    # Return value from fact pushed to be arg 2
        pushl   $.LC0                   # Push format string to be arg 1
        call    printf                  # Call the printf function
        addl    $8,%esp                 # Remove arguments from stack
#
        popl    %ebp                    # Restore frame pointer
        ret                             # Return to caller

#
        .data
.LC0:
        .string "The factorial of 10 is %d\n"
```

printf("The factorial of 10 is %d\n", fact(10));

```
#
# The factorial function itself
#
# ie fact(n)
#
        .text
        .align  4
        .globl  fact            # .globl also allows gdb to see label
fact:
        pushl   %ebp            # Save old frame pointer
        movl    %esp,%ebp       # Establish new frame pointer
#
        cmpl    $0, 8(%ebp)     # Test n against 0     EFLAGS
        jg      .L2             # Branch if n > 0
        movl    $1,%eax         # Return 1
        jmp     .L1             # Jump to code to return
#
.L2:
        movl    8(%ebp),%eax    # Get n
        subl    $1,%eax         # Compute (n - 1)
        pushl   %eax            # Push it to pass as argument
        call    fact            # Recursive call
        addl    $4,%esp         # Remove argument
#
        imull    8(%ebp),%eax   # Compute n * fact(n - 1)
#
.L1:                            # Result is in %eax
        popl    %ebp            # Restore frame pointer
        ret                     # Return to caller
```