The Intel 64 Architecture

CS 520
Dept. of Computer Science
Univ. of New Hampshire
Known generically as **x86-64**.

**Intel 64** is Intel's implementation.

**AMD 64** is AMD's implementation.

The design originated with AMD and was later adopted by Intel.

**Note:** This is not the Intel IA-64 architecture, which is Intel's other 64-bit architecture, for their Itanium processors.
Main Lecture Goal

Understand how function calls are supported on the Intel 64.

Recursion

Return address

Return values

Parameters

Necessary in order to understand how to implement garbage collectors, threads, etc.
Intel 64

64-bit addresses

64-bit integer registers
- rax, rbx, rcx, rdx, rdi, rsi and r8-r15.
- rsp - stack pointer
- rbp - frame pointer
- rip - 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
word - 16 bits
long - 32 bits
quardword - 64 bits

< Types
char
short
int
long
Stack grows from high address down to low address. RSP points to the top of stack.
Frames

Stack contains a series of frames, one for each function call.

Each frame contains:
- Return address
- Saved registers
- Local variables
- Parameters
- Temporary

rbp points to the top frame on the stack.
Integer parameters

1. rdi
2. rsi
3. rdx
4. rcx
5. r8
6. r9
7. ↓ on stack
return value

Integer return values are returned in $\text{gX}$.
Savings/restoring registers

calling function is responsible for saving and restoring \texttt{rax}, \texttt{rcx}, \texttt{rdx}, \texttt{rdi}, \texttt{rsi}, \texttt{rbp} if it needs the old value upon return.

\texttt{rbx}, \texttt{r12-r15}, \texttt{rsp}, \texttt{rbp} if it uses them.

\texttt{rbx}, \texttt{r12-r15}, \texttt{rsp}, \texttt{rbp} if it uses them.

\texttt{rbx}, \texttt{r12-r15}, \texttt{rsp}, \texttt{rbp} if it uses them. Saved upon entry & restored before return.
# x86-64 (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 8
.globl main

main:
    pushq  %rbp    # Save old frame pointer
    movq   %rsp,%rbp # Establish new frame pointer
    movq   $10,%rdi  # pass 10 as an argument
    call   fact  # Call factorial function
    movq   $.LC0,%rdi  # Pass format string as arg 1
    movq   $rax,%rsi  # Pass return value from fact as arg 2
    call   printf  # Call the printf function
    popq   %rbp  # Restore frame pointer
    ret  # Return to caller

.data
.LC0:
.string   "The factorial of 10 is %ld\n"

# The factorial function itself
# ie fact(n)

.text
.align 8
.globl fact  # .globl also allows gdb to see label

fact:
    pushq  %rbp    # Save old frame pointer
    movq   %rsp,%rbp # Establish new frame pointer
    subq   $8,%rsp   # Allocate one local
    cmpq   $0,%rdi  # Test n against 0
    jg   .L2        # Branch if n > 0
    movq   $1,%rax  # Return 1
    jmp   .L1       # Jump to code to return

.L2:
    movq   %rdi,-8(%rbp) # Save n into the local
    subq   $1,%rdi  # Compute (n - 1)
    call   fact  # Recursive call
    imulq  -8(%rbp),%rax # Compute n * fact(n - 1)

.L1:
    addq   $8,%rsp  # Result is in %rax
    popq   %rbp  # Deallocate the local
    ret  # Restore frame pointer
    ret  # Return to caller
The factorial function itself

```assembly
.text
.align 8
.globl fact

fact:
pushq $rbp
movq %rsp, %rbp
Subq $8, %rsp
# .globl also allows gdb to see label

cmpq $0, %rdi
jg .L2
# Save old frame pointer
# Establish new frame pointer
# Allocate one local

movq $1, %rax
# Test n against 0
# Branch if n > 0
jmp .L1
# Return 1
# Jump to code to return

.L2:
movq %rdi, -8(%rbp)
subq $1, %rdi
# Save n into the local
# Compute (n - 1)

imulq -8(%rbp), %rax
# Recursive call

.L1:
addq $8, %rsp
# Compute n * fact(n - 1)

lea
popq %rbp
# Result is in %rax
# Deallocate the local
# Restore frame pointer
# Return to caller
ret
```

---

- `text - instructions`
- `data - data`
- `flags - registers`
- `states - registers`
- `static - high level function`
- `default - visible`
- `asm - visible`
- `default - hidden`
rsp →

![Diagram]

rbp →

locals

rbp

rip

← call pushes
ret pops
x86-64 (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).

```assembly
.text
.align 8
.globl main
main:
pushq %rbp  # Save old frame pointer
movq %rsp,%rbp # Establish new frame pointer
movq $10,%rdi # pass 10 as an argument.
call fact # Call factorial function
movq $.LC0,%rdi # Pass format string as arg 1
movq %rax,%rsi # Pass return value from fact as arg 2
call printf # Call the printf function
popq %rbp # Restore frame pointer
ret # Return to caller
.data
.LC0:
.string "The factorial of 10 is $ld\n"
```

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