

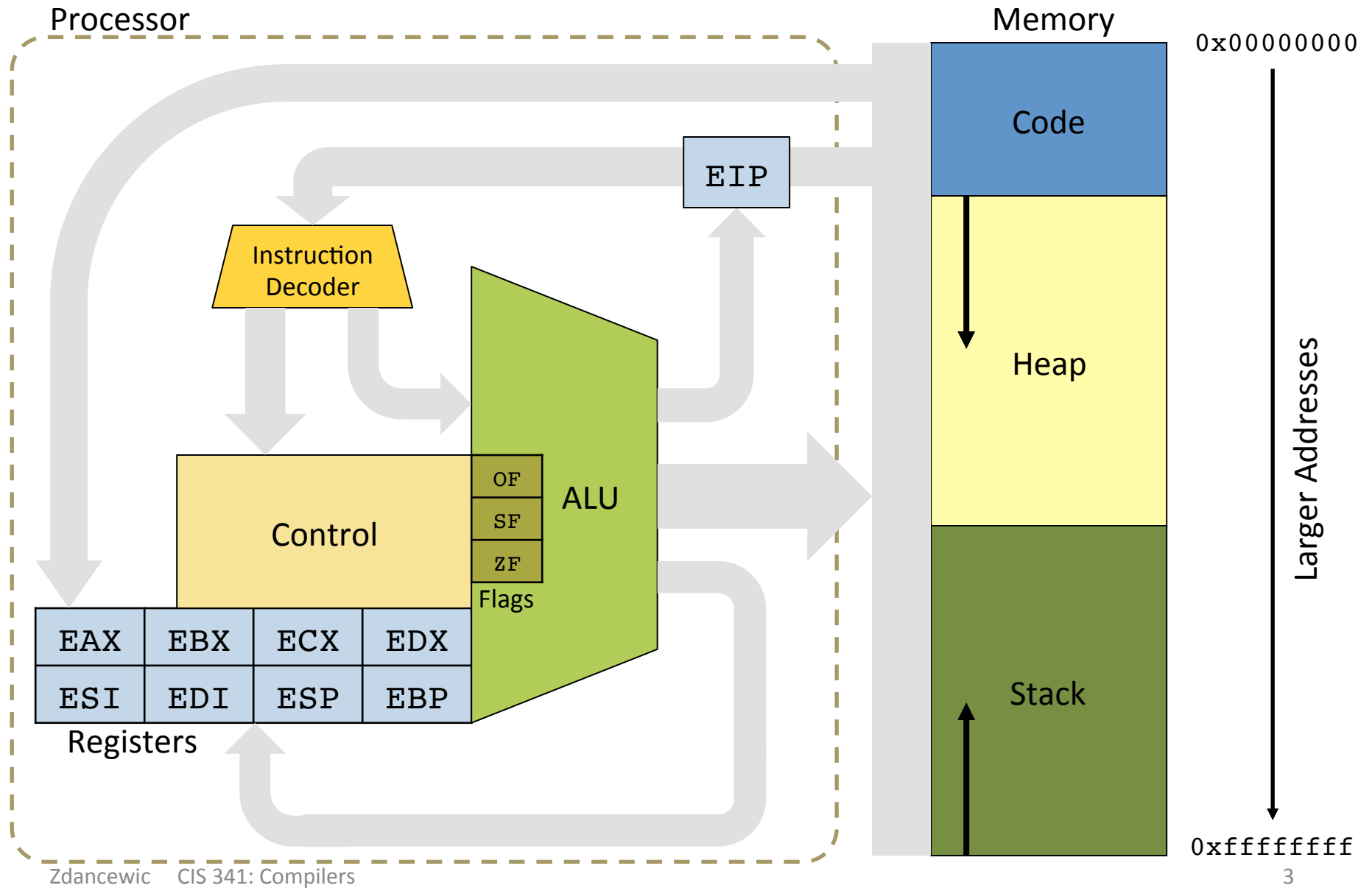
Lecture 4

CIS 341: COMPILERS

CIS 341 Announcements

- HW2: X86lite
 - Available on the course web pages.
 - Due: Thurs. Feb. 2nd at midnight
 - Pair-programming project

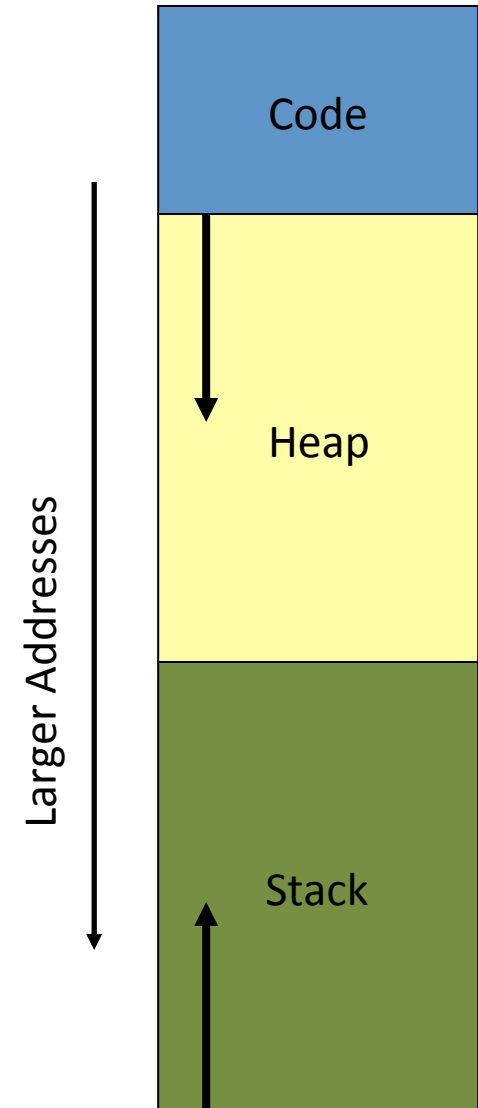
X86 Schematic



PROGRAMMING IN X86LITE

3 parts of the C memory model

- The code & data (or "text") segment
 - contains compiled code, constant strings, etc.
- The Heap
 - Stores dynamically allocated objects
 - Allocated via "malloc"
 - Deallocated via "free"
 - C runtime system
- The Stack
 - Stores local variables
 - Stores the return address of a function
- In practice, most languages use this model.



Local/Temporary Variable Storage

- Need space to store:
 - Global variables
 - Values passed as arguments to procedures
 - Local variables (either defined in the source program or introduced by the compiler)
- Processors provide two options
 - Registers: fast, small size (32 or 64 bits), very limited number
 - Memory: slow, very large amount of space (2 GB)
 - caching important
- In practice on X86:
 - Registers are limited (and have restrictions)
 - Divide memory into regions including the *stack* and the *heap*

Calling Conventions

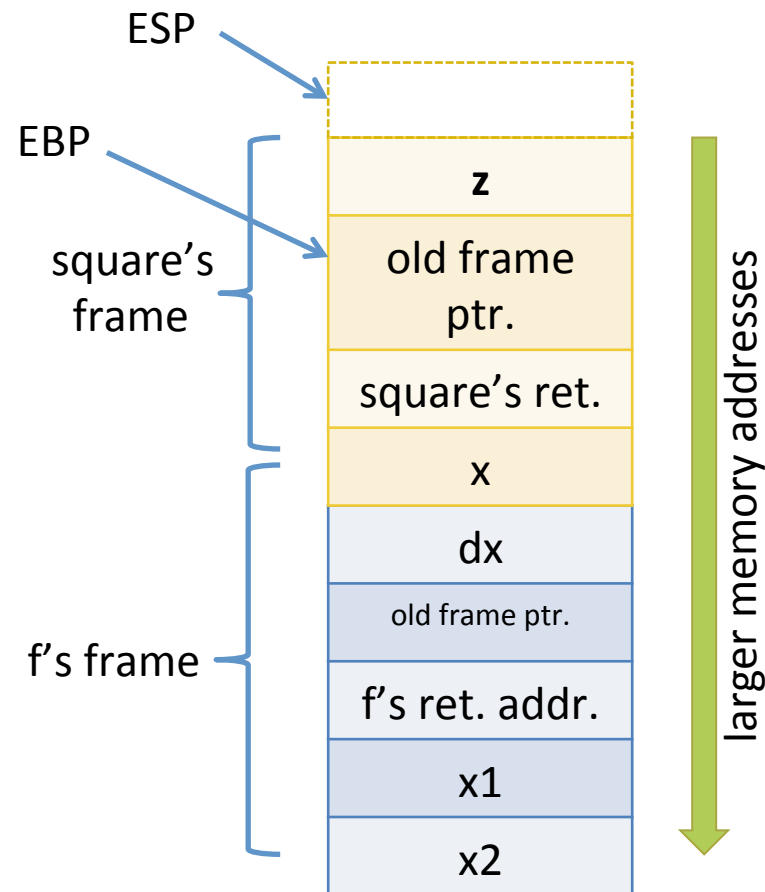
- Specify the locations (e.g. register or stack) of arguments passed to a function and returned by the function
- Designate registers either:
 - Caller Save – e.g. freely usable by the called code
 - Callee Save – e.g. must be restored by the called code
- Define the protocol for deallocating stack-allocated arguments
 - Caller cleans up
 - Callee cleans up (makes variable arguments harder)

32-bit cdecl calling conventions

- “Standard” on X86 for many C-based operating systems (i.e. almost all)
 - Still some wrinkles about return values (e.g. some compilers use **EAX** and **EDX** to return small values)
 - 64 bit allows for packing multiple values in one register
- Arguments are passed on the stack in right-to-left order
- Return value is passed in **EAX**
- Registers **EAX**, **ECX**, **EDX** are caller save
- Other registers are callee save
 - Ignoring these conventions will cause havoc (bus errors or seg faults)

Call Stacks: Example

- Use a stack to keep track of the return addresses:
 - `f` calls `g`, `g` calls `h`
 - `h` returns to `g`, `g` returns to `f`
- Stack frame:
 - Functions arguments
 - Local variable storage
 - Return address
 - Link (or “frame”) pointer



Call Stacks: Caller's protocol

- Function call:

$f(e_1, e_2, \dots, e_n);$

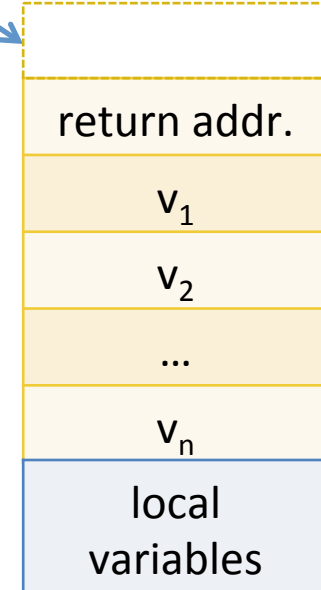
1. Save caller-save registers
2. Evaluate e_1 to v_1 , e_2 to v_2 , $\dots$, e_n to v_n
3. Push v_n to v_1 onto the top of the stack.
4. Use `call` to jump to the code for f
 - pushing the return address onto the stack.

- Invariant: returned value passed in `EAX`

- After call:

1. clean up the pushed arguments by popping the stack.
2. Restore caller-saved registers

ESP



larger memory addresses

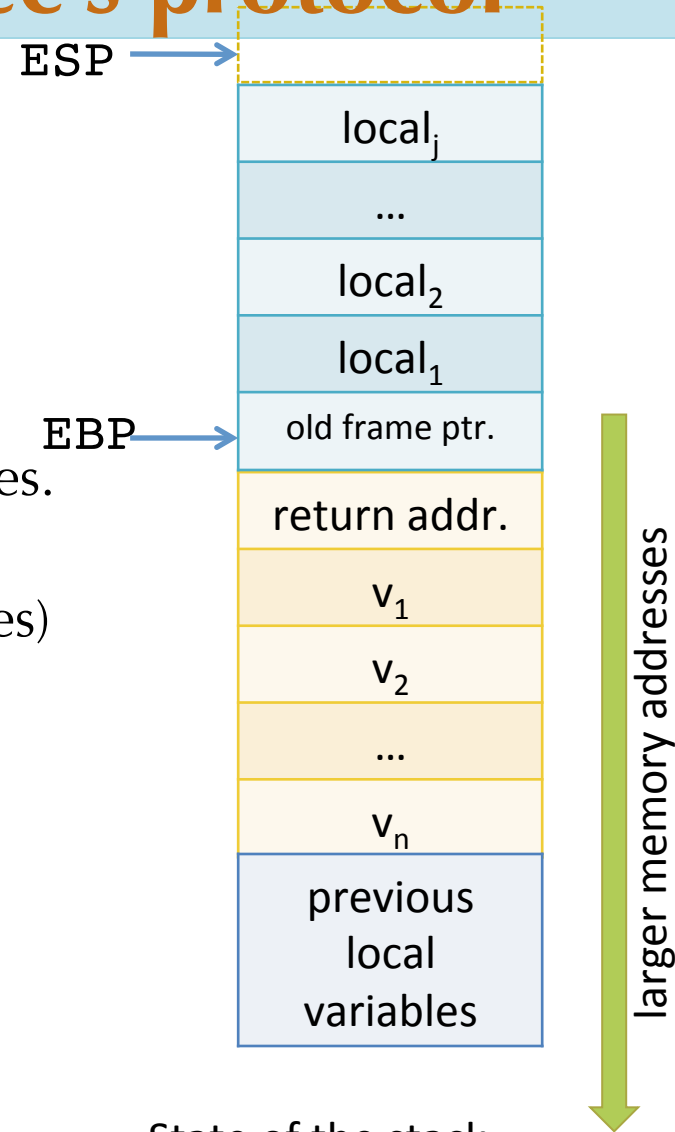
State of the stack
just after the Call
instruction:

Call Stacks: Callee's protocol

- On entry:
 1. Save old frame pointer
 - EBP is callee save
 2. Create new frame pointer
 - `Mov(Esp, Ebp)`
 3. Allocate stack space for local variables.

- Invariants: (assuming word-size values)
 - Function argument n is located at:
 $EBP + (1 + n) * 4$
 - Local variable j is located at:
 $EBP - j * 4$

- On exit:
 1. Pop local storage
 2. Restore EBP



State of the stack
after Step 3 of entry.

X86-64 SYSTEM V AMD 64 ABI

- More modern variant of C calling conventions
 - used on Linux, Solaris, BSD, OS X
- Callee save: `rbp`, `rbx`, `r12-r15`
- Caller save: all others
- Parameters 1 .. 6 go in: `rdi`, `rsi`, `rdx`, `rcx`, `r8`, `r9`
- Parameters 7+ go on the stack (in right-to-left order)
- Return value: in `rax`
- 128 byte "red zone" – scratch pad for the callee's data

See: handcoding.ml, [runtime.c](#)

DEMO: COMPILING EXPRESSIONS