Y86 Assembly

CS429H - Spring 2011

Christian Miller

The Y86 assembler

- The assembler is simple, just builds a memory map
- Normally, will just produce object code linearly
- Directives can modify that behavior
- You've already been over the Y86 instructions, we won't recap that here

Common directives

- .pos x: moves to address x in memory
- .align x: aligns to the next x byte boundary
- .long x: just dump value x in the memory map
- label: named labels can replace raw addresses

```c
int array[] = {0xd, 0xc0, 0xb00, 0xa000};

/* $begin sum-c */
int Sum(int *Start, int Count)
{
    int sum = 0;
    while (Count) {
                sum += *Start;
                Start++;
                Count--;
    }
    return sum;
}
/* $end sum-c */

int main()
{
    Sum(array, 4);
    return 0;
}
```

```
# Execution begins at address 0
    .pos 0
init:
    irmovl Stack, %esp          # Set up Stack pointer
    irmovl Stack, %ebp          # Set up base pointer
    jmp Main            # Execute main program

# Array of 4 elements
    .align 4
array:
    .long 0xd
    .long 0xc0
    .long 0xb00
    .long 0xa000

Main:
    irmovl $4,%eax
    pushl %eax  # Push 4
    irmovl array,%edx
    pushl %edx      # Push array
    call Sum    # Sum(array, 4)
    halt

    # int Sum(int *Start, int Count)
Sum:
    pushl %ebp              # Save old base pointer
    rrmovl %esp,%ebp        # Update base pointer
    mrmovl 8(%ebp),%ecx         # ecx = Start
    mrmovl 12(%ebp),%edx        # edx = Count
    irmovl $0, %eax             # sum = 0
    andl   %edx,%edx
    je     End
Loop:
    mrmovl (%ecx),%esi      # get *Start
    addl %esi,%eax          # add to sum
    irmovl $4,%ebx          #
    addl %ebx,%ecx          # Start++
    irmovl $-1,%ebx            #
    addl %ebx,%edx          # Count--
    jne    Loop             # Stop when 0
End:
    rrmovl %ebp,%esp        # Restore stack pointer
    popl %ebp               # Restore base pointer
    ret

    .pos 0x100
Stack:          # The stack goes here
```

Stack discipline

- Don didn't have time to get to this in class
- A stack is used to implement function calls, and the local storage for each function
- The stack is supported by the ISA
- Caller and callee need to agree on who does what when, or it all blows up

The x86 / Y86 stack

- Starts at the **top** of memory and grows **down**
- Each function has a **frame** where it stores its stuff
- Two registers keep track of the current stack frame:
  - %ebp is the base or frame pointer (start of the frame)
  - %esp is the stack pointer (end of the frame and top of the stack)

Stack instructions

- pushl rA
  - Decrement %esp by 4
  - Store contents of rA to memory at %esp
- popl rA
  - Read memory at %esp, store in rA
  - Increment %esp by 4

Calling instructions

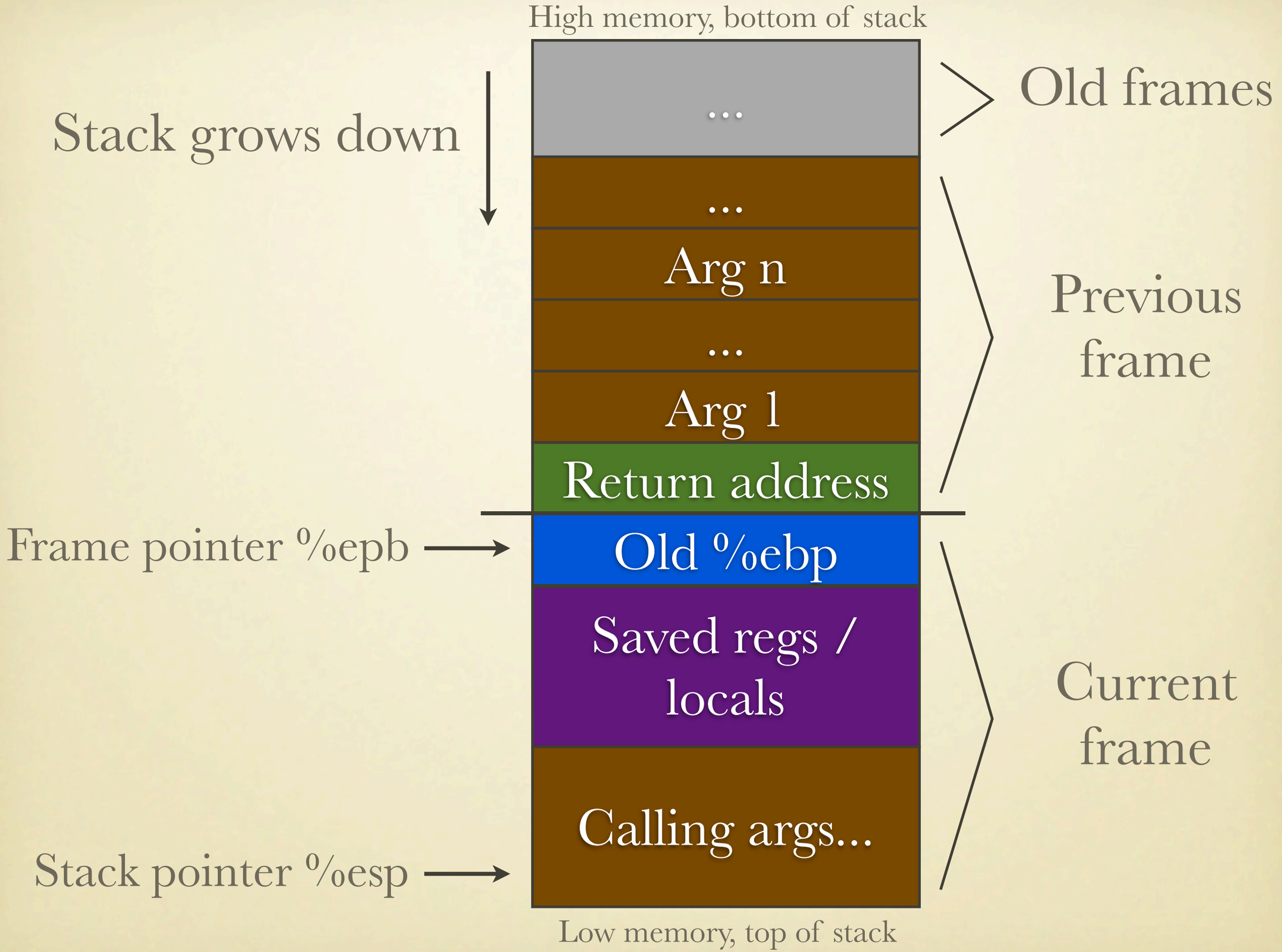
- call Dest
  - pushes next instruction onto stack
  - jumps to Dest
- ret
  - pops top value from stack
  - jumps to that location

Calling convention

- There are actually several, but this is most common
- Caller puts arguments on stack in reverse order
- Caller uses ‘call’ to push next instruction onto stack, then jump to the called function
- Callee pushes previous frame pointer %ebp onto stack, then overwrites it with %esp
- Callee pushes registers to save & local data, and does its business

Returning convention

- Callee restores any registers it saved
- Callee sets the stack pointer to its frame pointer
- Callee pops old frame pointer from the value it saved earlier on the stack
- Callee calls ‘ret’ to jump back to the caller
- Caller cleans up any arguments it pushed to the stack


High memory, bottom of stack
Stack grows down
...
Old frames
...
Arg n
...
Arg 1
Return address
Previous frame
Frame pointer %epb
Old %ebp
Saved regs / locals
Calling args...
Current frame
Stack pointer %esp
Low memory, top of stack

```
# Execution begins at address 0
    .pos 0
init:
    irmovl Stack, %esp         # Set up Stack pointer
    irmovl Stack, %ebp         # Set up base pointer
    jmp Main           # Execute main program

# Array of 4 elements
    .align 4
array:
    .long 0xd
    .long 0xc0
    .long 0xb00
    .long 0xa000

Main:
    irmovl $4,%eax
    pushl %eax  # Push 4
    irmovl array,%edx
    pushl %edx      # Push array
    call Sum    # Sum(array, 4)
    halt

    # int Sum(int *Start, int Count)
Sum:
    pushl %ebp             # Save old base pointer
    rrmovl %esp,%ebp       # Update base pointer
    mrmovl 8(%ebp),%ecx        # ecx = Start
    mrmovl 12(%ebp),%edx       # edx = Count
    irmovl $0, %eax            # sum = 0
    andl   %edx,%edx
    je     End
Loop:
    mrmovl (%ecx),%esi      # get *Start
    addl %esi,%eax          # add to sum
    irmovl $4,%ebx          #
    addl %ebx,%ecx          # Start++
    irmovl $-1,%ebx            #
    addl %ebx,%edx          # Count--
    jne    Loop             # Stop when 0
End:
    rrmovl %ebp,%esp       # Restore stack pointer
    popl %ebp              # Restore base pointer
    ret

    .pos 0x100
Stack:         # The stack goes here
```

Your assignment

- Write 3 simple programs in Y86 assembly
- You are given the C source code for them
- You have one week
- Download the code from the class labs webpage

```c
/* sum_list - Sum the elements of a linked list */
int sum_list(list_ptr ls)
{
    int val = 0;
    while (ls) {
        val += ls->val;
        ls = ls->next;
    }
    return val;
}

/* rsum_list - Recursive version of sum_list */
int rsum_list(list_ptr ls)
{
    if (!ls)
        return 0;
    else {
        int val = ls->val;
        int rest = rsum_list(ls->next);
        return val + rest;
    }
}

/* copy_block - Copy src to dest and return xor checksum of src */
int copy_block(int *src, int *dest, int len)
{
    int result = 0;
    while (len > 0) {
        int val = *src++;
        *dest++ = val;
        result ^= val;
        len--;
    }
    return result;
}
```

```c
/* linked list element */
typedef struct ELE {
    int val;
    struct ELE *next;
} *list_ptr;
```