CMSC 430 - 9 June 2026

Binary operations and the stack

- Fraud - Variables and let expressions, quick review
- Fraud - Binary primitive operations

CMSC 430 - 9 June 2026

Announcements

- Assignment 2 grades released
- Assignment 3 due Thursday by midnight
- Practice midterm released later today
- New quiz released, due by start of class tomorrow

Fraud

Learning objectives

Fraud

You should be able to answer (cont'd):

- What role does the stack play in computing the meaning of binary operations?
- What role does the compile-time environment play in the compilation of binary operations?
- How the ABI requirements for stack-alignment be achieved dynamically?

Fraud: adding binary operations

Concrete examples

```
(+ 3 4) ;=> 7
(- 4 3) ;=> 1
(< 3 4) ;=> #t
(= 3 4) ;=> #f
```

Fraud: adding binary operations

Abstract examples

```
(Prim2 '+ (Lit 3) (Lit 4))
(Prim2 '- (Lit 4) (Lit 3))
(Prim2 '< (Lit 3) (Lit 4))
(Prim2 '= (Lit 3) (Lit 4))
```

Fraud: adding binary operations

Updated semantics

```
;; type Value =
;; | Integer
;; | Boolean
;; | Character
;; | Eof
;; | Void
```

Unchanged!

Fraud: adding binary operations

Updated semantics

```
;; Expr Env -> Value { raises 'err }
(define (interp-e e r) ;; where r closes e
  (match e
    ;...
    [(Prim2 p e1 e2)
     (interp-prim2 p
                   (interp-e e1 r)
                   (interp-e e2 r))]))
```

Fraud: adding binary operations

Updated semantics

```
> (interp (parse '(+ 3 4)))
7
> (interp (parse '(- 4 3)))
1
> (interp (parse '(< 3 4)))
#t
> (interp (parse '(= 3 4)))
#f
```

Fraud: adding binary operations

Updated compiler

```
;; Expr CEnv -> Asm
(define (compile-e e c)  ;; where c closes e
  (match e
    ; ...
    [(Prim2 p e1 e2) (compile-prim2 p e1 e2 c)]))
```

Fraud: adding binary operations

Updated compiler

```
;; Op2 Expr Expr CEnv -> Asm
(define (compile-prim2 p e1 e2 c)
  (seq (compile-e e1 c)
       (compile-e e2 c)
       (match p
         ; ...
         ['+ (Add rax rax)])))
```

Fraud: adding binary operations

Updated compiler

```
> (exec (parse '(+ 1 1)))
2
```

Fraud: adding binary operations

Updated compiler

```
;; Op2 Expr Expr CEnv -> Asm
(define (compile-prim2 p e1 e2 c)
  (seq (compile-e e1 c)
       (Mov r8 rax)
       (compile-e e2 c)
       (match p
         ; ...
         ['+ (Add rax r8)])))
```

Fraud: adding binary operations

Updated compiler

```
> (exec (parse '(+ 1 1)))
2
> (exec (parse '(+ 1 2)))
3
```

Fraud: adding binary operations

Updated compiler

```
;; Op2 Expr Expr CEnv -> Asm
(define (compile-prim2 p e1 e2 c)
  (seq (compile-e e1 c)
       (Push rax)
       (compile-e e2 c)
       (Pop r8)
       (match p
         ; ...
         ['+ (Add rax r8)])))
```

Fraud: adding binary operations

Updated compiler

```
> (exec (parse '(+ 1 1)))
2
> (exec (parse '(+ 1 2)))
3
> (exec (parse '(+ 1 (+ 2 3))))
6
```

Fraud: adding binary operations

Updated compiler

```
#lang racket
(+ 1 (begin (write-byte 97) 2))
```

Breaks the ABI by calling a function
with an unaligned stack

No way to align once and for all

Calls can happen with any number of pushes

```
#lang racket
(let ((x (write-byte 97)))          ; 0 push
  (+ (begin (write-byte 98) 1)      ; 1 push
     (begin
       (write-byte 99)              ; 2 push
       2)))
```

Dynamically aligning the stack

Updated compiler

```
;; Asm
;; Dynamically pad the stack to be aligned for a call
(define pad-stack
  (seq (Mov r15 rsp)
       (And r15 #b1000)
       (Sub rsp r15)))

;; Asm
;; Undo the stack alignment after a call
(define unpad-stack
  (seq (Add rsp r15)))
```

Uses callee-saved register r15 to store stack change across call.
Works so long as the stack is either 8-byte or 16-byte aligned.
Since we always push 8-bytes at a time, this works for us.

For next time

- Read Hustle notes
- Take ELMS quiz