CS 271
Computer Architecture & Assembly Language

Lecture 16
Reverse Polish Notation (RPN)
Expression Evaluation
IA-32 Floating-Point Unit (FPU)
2/24/22, Thursday
Odds and Ends

• Final Project posted
  • Due Tuesday, March 15\textsuperscript{th} 11:59 pm

• Due 2/27 11:59 pm:
  • Program 5
  • Weekly Summary 8
  • Quiz 3
Lecture Topics:

• Reverse Polish Notation (RPN)
• Expression Evaluation
• IA-32 Floating-Point Unit (FPU)
Reverse Polish Notation (RPN)
Expression Evaluation
Reverse Polish Notation

• RPN
• Postfix form of expression
• Example
  • Infix: \( a + (b - c) \times (d + e) \)
  • Postfix (RPN): \( a \ bc- \ de+ \ abc-de+*+ \)
• Notice how operator precedence is preserved
• Notice how order of operands is preserved
• Notice how order of operators is **NOT** preserved
• RPN does not require parentheses
Conversion infix $\leftrightarrow$ postfix (RPN)

Binary Tree Method

- Fully parenthesize infix
- Don’t parenthesize postfix
- **Operands** are always in the original order
- Operators may appear in different order
Conversion $\text{infix} \leftrightarrow \text{postfix (RPN)}$

**Binary Tree Method**

1. Fully parenthesize the infix expression
   - Follow rules of operator precedence

2. Parse the expression left to right, constructing a binary tree
   - ( go left
   - Operand insert
   - Operator go up, insert, go right
   - ) go up

3. Post-order traversal gives RPN
Examples (infix $\rightarrow$ postfix)

\[
\begin{align*}
& (a + b) \times (c + d) + e \\
& ((a + b) \times (c + d)) + e \\
& ab + cd + * e + \\
& a \times b + (c \times d) \times e \\
& ((a \times b) + ((c \times d) \times e)) \\
& ab \times cd \times e + \\
& (a - b) \times (((c - d \times e) / f) / g) \times h \\
& ((a - b) \times ((c - (d \times e)) / f) / g)) \times h \\
& ab - cde - f / g / h * \\
\end{align*}
\]
Conversion postfix $\rightarrow$ infix

- Binary tree method
  - Diagram expression as a binary tree
    - Last operator is root
  - Do in-order traversal, parenthesizing each subtree
Example (post → infix)

• $ab + c + d^*$
  • $((a + b) + c) * d$

• $abcde + ** /$
  • $a / (b * (c * (d + e)))$

• $abcde * f + g - h / * +$
  • $a + (b * (((c + ((d * e) / f)) - g) / h))$
Evaluation of RPN Expressions

- Parse expression left to right, creating a stack of operands
  - Operand: push onto stack
  - Operator: pop 2 operands, perform operation, push result onto stack

- Single value remaining on the stack is value of expression
Examples (Evaluation of RPN Expressions)

• a = 5, b = 7, c = 4, d = 2, e = 3, f = 1, g = 6

• ab+cd*-
  • 4

• abcdefg++**++
  • 92

• abc+de*f/+g-*
  • 55
Using RPN in Programs

• Expression evaluation
• 0-address machine
  • E.g., Intel IA-32 FPU

• Example: Evaluate $a - b \times c$
  1. Convert to RPN $abc*-$
  2. Program
     - push a
     - push b
     - push c
     - mul
     - sub
a = b - (c + d)

- O-address: use RPN
- a = b c d + -

  push b
  push c
  push d
  add
  sub
  pop a
IA-32 Floating-Point Unit (FPU)
IA-32 Floating Point Processor (FPU)

- Runs in parallel with integer processor
- Circuits designed for fast computation on floating point numbers (IEEE format)
- Registers implemented as "pushdown" stack
- Usually programmed as a 0-address machine
  - Other instructions are possible
- CPU/FPU Exchange data through memory
  - Converts WORD and DWORD to REAL10
# Floating-point Unit Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IEEE 754 format</td>
</tr>
<tr>
<td>1</td>
<td>bit #79 : sign bit</td>
</tr>
<tr>
<td>2</td>
<td>bits #78 - #64 : biased exponent</td>
</tr>
<tr>
<td>3</td>
<td>bits #63 - #0 : normalized mantissa</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
Floating-point Unit Registers

• FPU is programmed as a “pushdown” stack
• If you push more than 8 values, the “bottom” of the stack will be lost
• Operations are defined for the “top” one or two registers
• Registers may be referenced by name ST(x)
Programming the FPU

• FPU Registers = ST(0) ... ST(7)
• ST=ST(0)=top of the stack
• ST(0) is implied when an operand is not specified
• Instruction Format:
  
  OPCODE
  OPCODE
  OPCODE     destination
  OPCODE     destination, source

  *Restrictions: One register must be ST(0)

• FINIT: initialize FPU register stack
  • Execute before any other FPU instructions!
Programming the FPU

• Note: (FPU instructions begin with ‘F’)
  • FSUB
  • FADD
  • Etc.

• To specify that a value being used is integer, use the special instructions that start with “FI”
  • FISUB
  • FIADD
  • Etc.
Sample Register Stack Opcodes

- **FLD MemVar**
  - Push ST(i) “down” to ST(i+1) for i = 0…6
  - Load ST(0) with MemVar

- **FST MemVar**
  - Move top of stack to memory
  - Leave result in ST(0)

- **FSTP MemVar**
  - Pop top of stack to memory
  - Move ST(i) “up” to ST(i-1) for i = 1…7
Sample FPU Opcodes

- Instructions use top of register stack as implied operand(s)
  - **FADD**: Addition (pop top two, add, push result)
  - **FSUB**: Subtraction
  - **FMUL**: Multiplication
  - **FDIV**: Division
  - **FDIVR**: Division (reverses operands)
  - **FSIN**: Sine (uses radians)
  - **FCOS**: Cosine (uses radians)
  - **FSQRT**: Square Root
  - **FABS**: Absolute Value
  - **FYL2X**: $Y \times \log_2(X)$ (X is in ST(0), Y is in ST(1))
  - **FYL2XP1**: $Y \times \log_2(X)+1$
Example

.data
varX REAL10 2.5
varY REAL10 -1.8
varZ REAL10 0.9
result REAL10 ?

.code
FINIT
FLD varX
FLD varY
FLD varZ
FMUL
FADD
FSTP result ; result = varX + (varY * varZ)
; etc.
Example

\[ \begin{array}{c}
\text{\(7 \rightarrow 6 \rightarrow \)} \\
\text{(6, 4.5) (2, 3.2)}
\end{array} \]

; Implementation of \((6.0 \times 2.0) + (4.5 \times 3.2)\)

; Note: RPN is \(6.0 \ 2.0 \times \ 4.5 \ 3.2 \times +\)

.data
array REAL10 6.0, 2.0, 4.5, 3.2
dotProduct REAL10 ?
0-Address Operations

```
array REAL10 6.0, 2.0, 4.5, 3.2

main PROC ; RPN is  6.0  2.0  *  4.5  3.2  *  +
  fld  array ; push 6.0 onto the stack
  fld  array+10 ; push 2.0 onto the stack
  fmul ; ST(0) = 6.0 * 2.0
  fld  array+20 ; push 4.5 onto the stack
  fld  array+30 ; push 3.2 onto the stack
  fmul ; ST(0) = 4.5 * 3.2
  fadd ; ST(0) = ST(0) + ST(1)
  fstp  dotProduct ; pop stack into memory
  exit
main ENDP
END main
```
Irvine’s Library

• ReadFloat: get keyboard input into ST(0)
• WriteFloat: display ST(0) contents in floating-point format

• Experiment!!