Programming Languages
(CS 550)

Mini Language Compiler

Jeremy R. Johnson
Introduction

• Objective: To illustrate how to map Mini Language instructions to RAL instructions. To do this in a systematic way that illustrates how to write a compiler to translate Mini Language programs to RAL programs. Show simple optimizations that can be used to reduce the number of instructions.

• Algorithm
  – Construct code for expressions, assignments, if, and while.
  – Concatenate statements in stmt-list
  – Allocate temporaries as needed
  – Keep track of variables, constants, and temporaries in Symbol Table
  – Use symbolic instructions and fill in absolute addresses (linking) when complete code has been constructed
A Random Access Machine

Control Unit

Program

AC = accumulator register

Memory

1
2
3
4
5
6
...
Instruction Set

- LDA X; Load the AC with the contents of memory address X
- LDI X; Load the AC indirectly with the contents of address X
- STA X; Store the contents of the AC at memory address X
- STI X; Store the contents of the AC indirectly at address X
- ADD X; Add the contents of address X to the contents of the AC
- SUB X; Subtract the contents of address X from the AC
- MUL X; Multiply the contents of address X to the contents of the AC
- JMP X; Jump to the instruction labeled X
- JMZ X; Jump to the instruction labeled X if the AC contains 0
- JMN X; Jump to the instruction labeled X if the contents of the AC is negative
- HLT; Halt execution
Memory Organization

- Constants
  - Num_Consts
- Prog. Variables
  - Num_Vars
  - get_temp()
- Temp. Variables
  - Num_Temps
Symbolic Instructions

- Addresses and labels can be symbolic names
- Symbolic names are mapped to actual addresses during linking

Example:
- LD x
- ST z
- ADD y
- JMP L

Linked code with (x=100, y =110, z = 105, L = 20)
- LD 100
- ST 105
- ADD 110
- JMP 20
Symbol Table

• Map from identifiers → Symbol table entries

• Symbol table entries contain: address [may be unknown]

• Indicate whether entry is a constant, variable, temporary or label
Expressions

expr → expr₁ op expr₂

Code₁ ; result stored in t₁
Code₂ ; result stored in t₂
LD t₁ ; load result of exp₁
OP t₂ ; apply op to result of exp₂ and result of exp₁
ST t₃ ; store result of exp₁ op exp₂
Expressions

expr $\rightarrow$ NUMBER

; check to see if NUMBER in symbol table,
; otherwise add to symbol table

LD NUMBER ; load constant from constant table
ST $t_n$ ; next available temporary
Expressions

\[ expr \rightarrow IDENT \]

; check to see if IDENT in symbol table
; otherwise add to symbol table

LD IDENT ; load constant from constant table
ST \( t_n \) ; next available temporary
Assignment

assign_stmt → IDENT = expr

; check to see if IDENT in symbol table
; otherwise add to symbol table

Code
LD t
ST IDENT
Conditional Statements

if_stmt → if expr then S_1 else S_2 fi

Code_e ; result stored in t
LD t ;
JMN L1
Code_1
JMP L2
L1: Code2
L2:
While Statements

while_stmt → while expr do S od

L1: Code_e ; result stored in t
    LD t  ;
    JMN L2
    Code_S
    JMP L1
L2:
Statement List

\[ \text{stmt-list} \rightarrow \text{stmt; stmt-list} \mid \text{stmt} \]

code_1
code_2
...
code_n
Example

\[
n := 0 - 5; \\
\text{if } n \text{ then } i := n \text{ else } i := 0 - n \text{ fi; } \\
fact := 1; \\
\text{while } i \text{ do } fact := fact \times i; \ i := i - 1 \text{ od}
\]
Example

\[ n := 0 - 5; \]

LD ZERO
ST T1
LD FIVE
ST T2
LD T1
SUB T2
ST T3
LD T3
ST n
Example

if n then i := n else i := 0 - n fi;

L1:  LD ZERO
     LD n
     ST T6
     ST T4
     LD T4
     LD T7
   JMN L1
   LD n
   ST T8
   LD T6
   ST T5
   ST T7
   LD T5
   ST T8
   LD T8
   ST i
   JMP L2

L2:
Example

fact := 1;

LD ONE
ST T9
LD T9
ST fact
Example

while i do
  fact := fact * i; i := i - 1
od

L3:  LD i
    ST T10
    JMN L4
    LD fact
    ST T11
    LD i
    ST T12
    LD T11
    MUL T12
    ST T13
    LD T13
    ST fact
    LD i
    ST T14
    LD T11
    MUL T12
    ST fact
    LD i
    ST T14
    LD ONE
    ST T15
    LD T14
    SUB T15
    ST T16
    LD T16
    ST i
    JMP L3

L4:
Complete Example
(concatenate and append HLT)

| L1: LD ZERO          | L2: LD ONE          | L3: LD i            | L4: HLT
|--------------------|--------------------|--------------------|------
<p>| LD ZERO            |                    |                    |      |
| ST T1              | ST T6              | ST T10             |      |
| LD FIVE            | LD n               | JMN L4             |      |
| ST T2              | ST T7              | LD fact            |      |
| LD T1              | LD T6              | ST T11             |      |
| SUB T2             | SUB T7             | LD i               |      |
| ST T3              | ST T8              | ST T12             |      |
| LD T3              | LD T8              | LD T11             |      |
| ST n               | ST i               | MUL T12            |      |
| LD n               |                    |                    |      |
| ST T4              |                    |                    |      |
| LD T4              |                    |                    |      |
| JMN L1             |                    |                    |      |
| LD n               |                    |                    |      |
| ST T5              |                    |                    |      |
| LD T5              |                    |                    |      |
| ST i               |                    |                    |      |
| JMP L2             |                    |                    |      |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
<th>addr</th>
<th>Type</th>
<th>Value</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZERO</td>
<td>0</td>
<td>const</td>
<td>T8</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>FIVE</td>
<td>5</td>
<td>const</td>
<td>ONE</td>
<td>1</td>
<td>const</td>
<td>?</td>
</tr>
<tr>
<td>n</td>
<td>u</td>
<td>var</td>
<td>T9</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>T1</td>
<td>u</td>
<td>temp</td>
<td>T10</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>T2</td>
<td>u</td>
<td>temp</td>
<td>T11</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>T3</td>
<td>u</td>
<td>temp</td>
<td>T12</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>T4</td>
<td>u</td>
<td>temp</td>
<td>T13</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>T5</td>
<td>u</td>
<td>temp</td>
<td>T14</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>T6</td>
<td>u</td>
<td>temp</td>
<td>T15</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>T7</td>
<td>u</td>
<td>temp</td>
<td>T16</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
</tbody>
</table>
Symbol Table and Label Summary

Num_Vars = 3
Num_Consts = 3
Num_Temps = 16

Constants
  ZERO -> addr 1
  FIVE  -> addr 2
  One   -> addr 3

Variables
  n     -> addr 4
  i     -> addr 5
  fact  -> addr 6

Temporaries
  T1    -> addr 7
  T2    -> addr 8
  ...  
  T16   -> addr 22

Labels
  L1 = 19
  L2 = 28
  L3 = 32
  L4 = 54
# Linked Example

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L1: LD 1</td>
<td>L2: LD 3</td>
<td>L3: LD 5</td>
<td>L4: HLT</td>
</tr>
<tr>
<td>ST 12</td>
<td>ST 15</td>
<td>ST 16</td>
<td>ST 20</td>
</tr>
<tr>
<td>LD 4</td>
<td>LD 14</td>
<td>JMN 53</td>
<td>LD 3</td>
</tr>
<tr>
<td>ST 13</td>
<td>ST 5</td>
<td>LD 6</td>
<td>ST 21</td>
</tr>
<tr>
<td>LD 12</td>
<td>LD 19</td>
<td>ST 17</td>
<td>LD 20</td>
</tr>
<tr>
<td>SUB 13</td>
<td>SUB 21</td>
<td>LD 5</td>
<td>ST 22</td>
</tr>
<tr>
<td>LD 14</td>
<td>ST 5</td>
<td>ST 18</td>
<td>LD 22</td>
</tr>
<tr>
<td>ST 10</td>
<td>ST 17</td>
<td>LD 17</td>
<td>ST 5</td>
</tr>
<tr>
<td>LD 10</td>
<td>MUL 18</td>
<td>JMP 32</td>
<td>L4: HLT</td>
</tr>
<tr>
<td>JMN 19</td>
<td>LD 18</td>
<td>ST 6</td>
<td></td>
</tr>
<tr>
<td>LD 4</td>
<td>ST 12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST 11</td>
<td>LD 12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD 11</td>
<td>SUB 13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST 5</td>
<td>LD 13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP 28</td>
<td>LD 14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Optimizations

- Peephole optimization
  - Remove ST immediately followed by LD

- Commute \((\text{expr}_1,\text{expr}_2)\) in \(\text{expr} \rightarrow \text{expr}_1 \text{ op } \text{expr}_2\) to allow additional peephole optimizations

- Constant folding

- Common subexpression elimination
### Complete Example
*(after peephole optimization)*

<table>
<thead>
<tr>
<th>L1: LD ZERO</th>
<th>L2: LD ONE</th>
<th>L3: LD i</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD ZERO</td>
<td>ST T6</td>
<td>ST T10</td>
</tr>
<tr>
<td>ST T1</td>
<td>LD n</td>
<td>ST T10</td>
</tr>
<tr>
<td>LD FIVE</td>
<td>ST T7</td>
<td>JMN L4</td>
</tr>
<tr>
<td>ST T2</td>
<td>LD T6</td>
<td>LD fact</td>
</tr>
<tr>
<td>LD T1</td>
<td>SUB T7</td>
<td>ST T11</td>
</tr>
<tr>
<td>SUB T2</td>
<td>ST T8</td>
<td>LD i</td>
</tr>
<tr>
<td>ST T3</td>
<td>LD T8</td>
<td>ST T12</td>
</tr>
<tr>
<td>LD T3</td>
<td>ST i</td>
<td>LD T11</td>
</tr>
<tr>
<td>ST n</td>
<td>ST i</td>
<td>MUL T12</td>
</tr>
<tr>
<td>LD n</td>
<td>ST T9</td>
<td>ST T13</td>
</tr>
<tr>
<td>ST T4</td>
<td>LD T9</td>
<td>LD T13</td>
</tr>
<tr>
<td>LD T4</td>
<td>ST fact</td>
<td>ST fact</td>
</tr>
<tr>
<td>JMN L1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST T5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD T5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP L2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Complete Example
(after peephole optimization)

LD ZERO
ST T1
LD FIVE
ST T2
LD T1
SUB T2
JMN L1
LD n
ST i
JMP L2
L1:  LD ZERO
     ST T6
     LD n
     ST T7
     LD T6
     SUB T7
     ST i
L2:  LD ONE
     ST fact
L3:  LD i
     ST T10
     JMN L4
     LD fact
     ST T11
     LD i
     ST T12
     MUL T12
     ST fact
     ST T14
     LD ONE
     ST T15
     LD T14
     SUB T15
     ST i
     JMP L3
L4:  HLT

38 vs. 54 instructions
Supporting Procedures

- **Fully static environment**
  - No recursion
  - Activation record
    - Parameters
    - Local variables (keep count)
    - Return address (indirect jump needed)
    - Can be statically allocated

- **Dynamic environment**
  - Allow recursion
  - Call stack (dynamic allocation)
  - Indirect load and store needed
Memory Organization

Constants

Global

Prog. Variables

Global

Temp. Variables

Activation Records

Call Stack

Global

Temp. Variables

Global

Prog. Variables

Constants
Program Memory Organization

<table>
<thead>
<tr>
<th>Procedures</th>
<th>Procedure Entry in Function Table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Number of parameters</td>
</tr>
<tr>
<td></td>
<td>• Number of local/temp variables</td>
</tr>
<tr>
<td>P1</td>
<td>• Starting address</td>
</tr>
<tr>
<td>P2</td>
<td>• Number of instructions</td>
</tr>
<tr>
<td>P3</td>
<td>• Need to know starting address of main program</td>
</tr>
<tr>
<td>Main Program</td>
<td></td>
</tr>
</tbody>
</table>
Activation Record

- Parameters
- Local Variables
- Temp. Variables
- Return Address

Frame Pointer

For call stack

Stack Pointer
Example: fact(n)

define
    proc(n)
        i := n;
        fact := 1;
        while i do
            fact := fact * i;
            i := i - 1
        od;
        return := fact
    end
fact(n)

LD n  
ST T1  
LD T1  
ST i  
LD ONE  
ST T2  
LD T2  
ST fact  

L1:  
LD i  
ST T3  
JMN L2  
LD fact  
ST T4  
LD i  
ST T5  
LD T4  
MUL T5  
ST T6  
LD T6  
ST fact  

LD i  
ST T7  
LD ONE  
ST T8  
LD T7  
SUB T8  
ST T9  
LD T9  
ST i  
JMP L1  

L2:  
LD fact  
ST T10  
LD T10  
ST return
### Activation Record

<table>
<thead>
<tr>
<th>FP</th>
<th>SP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td></td>
</tr>
<tr>
<td>fact</td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td></td>
</tr>
<tr>
<td>T7</td>
<td></td>
</tr>
<tr>
<td>T8</td>
<td></td>
</tr>
<tr>
<td>T9</td>
<td></td>
</tr>
<tr>
<td>T10</td>
<td></td>
</tr>
<tr>
<td>return</td>
<td>ret. addr.</td>
</tr>
</tbody>
</table>

#### Accessing AR

- LD n ⇔ LDI FP
- ST n ⇔ STI FP
- LD i ⇔ LD FP
- ADD ONE
  - ST FPB
  - LDI FPB
- ⇔ LDO FP[1]
- ST i ⇔ STO FP[1]
- LD Tj ⇔ LDO FP[j+Num_Param+Num_Vars]

---

33
Calling Sequence

Initiate call
1. Create activation record
   1. Update FP and SP
2. Store parameters in activation record
3. Store return address (RA)
4. Jump to starting address of procedure code
   1. Introduce call instruction (can place RA relative to SP)

Return from call
1. Store return value in activation record (when return is assigned)
2. Jump to RA
   1. Introduce ret instruction (jmp indirect)
3. Retrieve return value from activation record
4. Update FP and SP