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

AC = accumulator register

AC = accumulator register
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
- Prog. Variables
- Temp. Variables

- Num_Consts
- Num_Vars
- get_temp()
- 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 an 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 \text{NUMBER}

; check to see if \text{NUMBER} in symbol table,
; otherwise add to symbol table

LD \text{NUMBER} ; \text{load constant from constant table}

ST \ t_n ; \text{next available temporary}
Expressions

expr → 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

$$\text{ifStmt} \rightarrow \text{if expr then } S_1 \text{ else } S_2 \text{ fi}$$
$$\Leftrightarrow \text{if expr > 0 then } S_1 \text{ else } S_2 \text{ fi}$$

\[\text{Code}_e \quad ; \text{result stored in } t\]
\[\text{LD } t \quad ;\]
\[\text{JMN L1 } \quad \text{Jump if } t \leq 0 \text{ then}\]
\[\text{JMZ L1}\]
\[\text{Code}_1\]
\[\text{JMP L2}\]
\[\text{L1: Code}_2\]
\[\text{L2:}\]
While Statements

while_stmt → while expr do S od
⇔ while expr > 0 then S od

L1: Code_e ; result stored in t
    LD t ;
    JMN L2 ; jump if t ≤ 0
    JMZ L2
    Code_S
    JMP L1
L2:
Statement List

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

code_1

code_2

\[ \ldots \]

code_n
Example

n := 0 - 5;
if n then i := n else i := 0 - n fi;
fact := 1;
while i do fact := fact * i; i := i - 1 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;

LD n
ST T4
LD T4
JMN L1
JMZ L1
LD n
ST T5
LD T5
ST i
JMP L2

L1:  LD ZERO
     ST T6
     LD n
     ST T7
     LD T6
     SUB T7
     ST T8
     LD T8
     ST i

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
    LD T10
    JMN L4
    JMZ L4
    LD fact
    ST T11
    LD i
    ST T12
    LD T11
    MUL T12
    ST T13

LD T13
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

LD ZERO
ST T1
LD FIVE
ST T2
LD T1
SUB T2
ST T3
LD T3
ST n
LD n
ST T4
LD T4
JMN L1
JMZ L1
LD n
ST T5
LD T5
ST i
JMP L2

L1:  LD ZERO   L2:  LD ONE
     ST T6                         ST T9
     LD n                         ST T9
     ST T7                         ST T9
     LD T6                       L2:  LD ONE
     LD T6                      MUL T12
     SUB T7                      ST T12
     ST i                       LD T11
     SUST i                      ST T12
          ST i                  LD T11
     JMP L3                      ST i

L3:  LD i
     ST T10
     JMN L4
     ST T11
     JMN L4
     LD fact
     ST T12
     MUL T12
     ST T13
     ST T13
     ST fact

L4:  HLT
     ST T14
     LD ONE
     ST T15
     LD T14
     SUB T15
     ST T16
     LD T16
     ST i
     JMP L3

LD i
## Symbol Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
<th>addr</th>
<th>T8</th>
<th>Value</th>
<th>Type</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZERO</td>
<td>0</td>
<td>const</td>
<td>?</td>
<td>ONE</td>
<td>1</td>
<td>const</td>
<td>?</td>
</tr>
<tr>
<td>FIVE</td>
<td>5</td>
<td>const</td>
<td>?</td>
<td>T9</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>n</td>
<td>u</td>
<td>var</td>
<td>?</td>
<td>fact</td>
<td>u</td>
<td>var</td>
<td>?</td>
</tr>
<tr>
<td>T1</td>
<td>u</td>
<td>temp</td>
<td>?</td>
<td>T10</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>T2</td>
<td>u</td>
<td>temp</td>
<td>?</td>
<td>T11</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>T3</td>
<td>u</td>
<td>temp</td>
<td>?</td>
<td>T12</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>T4</td>
<td>u</td>
<td>temp</td>
<td>?</td>
<td>T13</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>T5</td>
<td>u</td>
<td>temp</td>
<td>?</td>
<td>T14</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>i</td>
<td>u</td>
<td>var</td>
<td>?</td>
<td>T15</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>T6</td>
<td>u</td>
<td>temp</td>
<td>?</td>
<td>T16</td>
<td>u</td>
<td>temp</td>
<td>?</td>
</tr>
<tr>
<td>T7</td>
<td>u</td>
<td>temp</td>
<td>?</td>
<td></td>
<td></td>
<td></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
- ...  -> addr 8
- T16  -> addr 22

#### Labels
- L1 = 20
- L2 = 29
- L3 = 33
- L4 = 55
### Linked Example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>L1: Instruction</th>
<th>L2: Instruction</th>
<th>L3: Instruction</th>
<th>L4: Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD 1</td>
<td>LD 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST 7</td>
<td>ST 12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD 2</td>
<td>ST 13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST 8</td>
<td>LD 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD 7</td>
<td>ST 16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB 8</td>
<td>LD 12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST 9</td>
<td>ST 17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD 9</td>
<td>LD 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST 4</td>
<td>ST 18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD 4</td>
<td>LD 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST 10</td>
<td>LD 17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD 10</td>
<td>ST 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMN 20</td>
<td>ST 15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMZ 20</td>
<td>ST 19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD 4</td>
<td>LD 19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST 11</td>
<td>ST 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD 11</td>
<td>JMN 55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST 5</td>
<td>JMZ 55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP 29</td>
<td>LD 20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ST 20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LD 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ST 21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LD 22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ST 22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ST 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>JMP 33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>L4: HLT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Optimizations

- Peephole optimization
  - Remove LD immediately following by ST

- 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)

LD FIVE
ST T2
LD ZERO
ST T1
SUB T2
ST T3
ST n
ST T4
JMN L1
JMZ L1
LD n
ST T5
ST i
JMP L2

L1:  LD ZERO
     ST T6
     LD n
     ST T7
     LD T6
     SUB T7
     ST T8
     ST i

L2:  LD ONE
     ST T9
     ST fact

L3:  LD i
     ST T10
     JMN L4
     JMZ L4
     LD i
     ST T12
     LD fact
     ST T11
     MUL T12
     ST T13
     ST fact
     L4: HLT

46 vs. 56 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)
  - Use stack pointer (sp) and frame pointer (fp) access stack
  - Indirect load and store needed
Memory Organization

- Static
  - Constants
  - Global
  - Global
  - Global
  - Global
  - Temp. Variables
  - Temp. Variables
  - Activation Records

- Dynamic
  - FP, SP & scratch
  - Constants
  - Global
  - Global
  - Global
  - Prog. Variables
  - Prog. Variables
  - Temp. Variables
  - Temp. Variables
  - Call Stack
Program Memory Organization

Procedure Entry in Function Table

<table>
<thead>
<tr>
<th>Procedures</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>Main Program</th>
</tr>
</thead>
</table>

- Number of parameters
- Number of local/temp variables
- Starting address
- Number of instructions

- Need to know starting address of main program
Example: fact(n)

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

```
LD n  L1: LD i  LD i
ST T1  ST T3  ST T7
LD T1  JMN L2  LD ONE
ST i  JMZ L2  ST T8
LD ONE  LD fact  LD T7
ST T2  ST T4  SUB T8
LD T2  LD i  ST T9
ST fact  ST T5  LD T9
LD T4  ST i  JMP L1
MUL T5  L2: LD fact  L2: LD fact
ST T6  ST T10  ST T10
LD T6  LD T10  ST return
ST fact  ST fact
```

Note that addressing (LD/ST) different than main program.
If main were a function the code would be uniform.
Activation Record

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]
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)
   2. Can compute RA from PC

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