IN2009

Language Processors

Session 7

Translation to Intermediate Representation

Christian Cooper

Session Plan

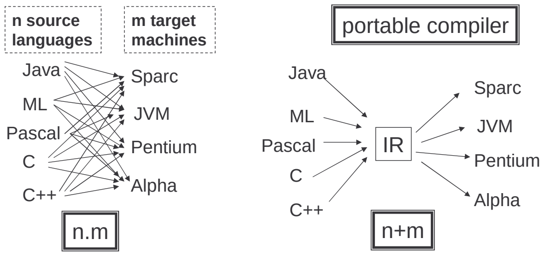
Session 7: Translation to intermediate representation

- Intermediate representation
- Assembly language - a recap
- Why use IR
- Definition of an IR using trees
- Example translations
- See book for while-loops, for-loops, functions, declarations

Intermediate representation

- Semantic analysis
  - converts abstract syntax to abstract machine code (an intermediate rep)
- why an intermediate representation…?

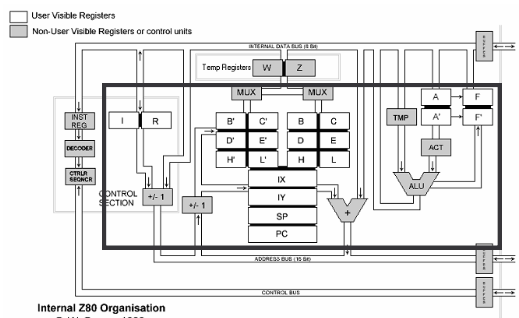
Intermediate representation



Assembly Language - a recap

- We've *briefly* considered Processor architecture and memory.
- We've *briefly* considered two architectures
  - Z80A
  - MIPS
- Details of the register sets of each...

Zilog Z80 (8-bit)



Internal Z80 Organisation
C. W. Cooper 1999

MIPS R3000 (32-bit)

Registers

Name	Number	Use	Callee must preserve?
$zero	$0	constant 0	N/A
$at	$1	assembler temporary	no
$v0–$v1	$2–$3	Values for function returns and expression evaluation	no
$a0–$a3	$4–$7	function arguments	no
$t0–$t7	$8–$15	temporaries	no
$s0–$s7	$16–$23	saved temporaries	**yes**
$t8–$t9	$24–$25	temporaries	no
$k0–$k1	$26–$27	reserved for OS kernel	no
$gp	$28	global pointer	**yes**
$sp	$29	stack pointer	**yes**
$fp	$30	frame pointer	**yes**
$ra	$31	return address	N/A

Microprocessor without Interlocked Pipeline Stages

Intermediate representation

- We represent the immediate output of the compilation using an *intermediate language* - an abstract machine language
  - Express target machine operations, but without machine-specific details
  - Source-language independent

Intermediate representation

- Compiler
  - front end: lexical analysis, parsing, semantic analysis
  - back end:
    - optimisation of IR (rewrite IR so as to improve execution speed)
    - translation to real machine language
    - (in case of Java, to another abstract machine language JVM)
- Many IRs
  - Appel uses *simple expression trees*

Real life IR and ILs

- A wide variety - for example gcc supports (amongst others):
  - RTL (register transfer language), GENERIC (tree-based), GIMPLE (a *static single assignment* language).
- Eiffel uses a simplified form of C.
- Java produces byte code.
- Microsoft .NET uses CIL.
- Other languages operate on pseudo-assembler or generic trees.

Intermediate representation

- Any good representation:
  - must be convenient for semantic analysis phase to produce.
  - must be convenient to translate to real (or virtual) machine language for target machines.
  - must have simple meaning for each construct that leads to simple operations on the IR to rewrite parts of it for optimisation etc.

Intermediate representation

- In any IR
  - individual components describe simple operations on the abstract machine represented by the IR instructions
  - each element of the complex abstract syntax is translated into a set of simple IR abstract machine instructions
  - groups of IR instructions will be grouped and regrouped to form real machine instructions

IR: tree expression operators

- `CONST(i)`
  - integer constant
- `NAME(n)`
  - symbolic constant (an assembly lang label)
- `TEMP(t)`
  - abstract register...infinite number!
- `BINOP(o,e1,e2)`
  - Where o = PLUS, MINUS, MUL, DIV, AND, OR, XOR, LSHIFT, RSHIFT, ARSHIFT

IR: tree expression operators

- `MEM(e)`
  - contents of wordSize bytes starting at addr e (means “store” if left child of move, else “fetch”)
- `CALL(f,l)`
  - procedure call, applies f to list l
- `ESEQ(s,e)`
  - eval s for side-effects, eval e for result

IR: statements

perform side effects & control flow

- `MOVE(TEMP t,e)`
  - eval e & move into temp t
- `MOVE(MEM(e1,k),e2)`
  - eval e1 (⇒ addr a); eval e2 & store results into k bytes of mem starting at a
- `EXP(e)`
  - eval e & discard the result

No provision for procedure and & function defs - just body of each function

IR: statements

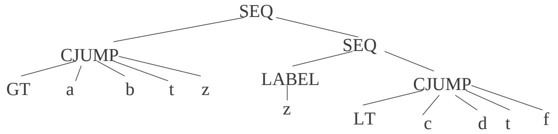
- `JUMP(e,labs)`
  - transfer control to addr e; labs specifies list of all poss locations e can eval to
- `CJUMP(o,e1,e2,t,f)`
  - eval e1 then e2, compare result with relational op o; if true jump to t else to f
- `SEQ(s1,s2)`
  - ???
- `LABEL(n)`
  - defines const n to be current machine code address

Example translation

a > b | c < d *translates to*

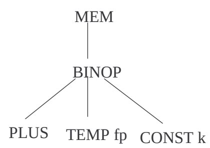
```
SEQ(CJUMP(GT,a,b,t,z),
    SEQ(LABEL z,
        CJUMP(LT,c,d,t,f)))
```

with t,f *labels*



Simple variables translation

- Simple variable v in current procedure or function stack frame
  - k: offset of v in frame
  - `TEMP fp`: frame pointer register



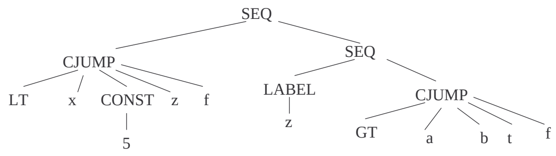
MEM(BINOP(PLUS,TEMP fp, CONST k))

Conditionals

- Use `CJUMP`
  - x < 5 translates to **`CJUMP(LT, x, CONST(5), t, f)`**
  - for labels `t`, `f`
- if x < 5 then a > b else 0

Conditionals

```
SEQ(CJUMP(LT,x,CONST(5),z,f),
  SEQ(LABEL z,
     CJUMP(GT,a,b,t [pick up val of a>b],f  [pick up val 0])))
```



What you should do now

- See book for other translations
  - while-loops
  - for-loops
  - functions
  - declarations
- Complete the May 2006 exam paper before next week!

Schedule

- ***Module Summary & Exam Revision***
- Monday 2nd April, 2007
  - 12:00 - 13:50
  - CM383