City University London

IN2009

**Language Processors**

Session 6

# Intermediate Representation
# Introducing TPL

Igor Siveroni

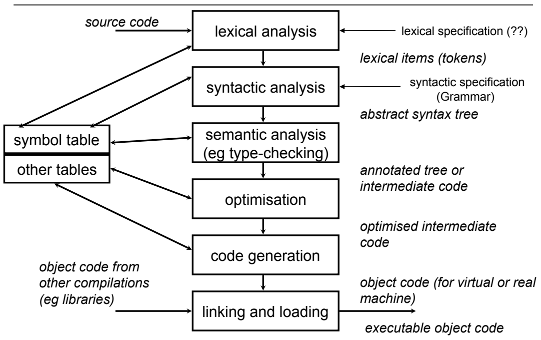
## Session Plan

Session 6: Intermediate Representation

- Why
- The Target Programming Language (TPL)
- Syntax of a subset of TPL
- Semantics of TPL
- Examples



## Language processing






## Intermediate Representation

We must decide whether to generate some intermediate representation (IR) or to generate target machine code directly. We choose to generate IR:

- The target machine is abstracted to some virtual machine, thus separating high-level operations from their possible low-level machine dependent realisations.
- Target machine dependencies are isolated to the actual code generation routines.
- Simpler optimisation can be done at the IR level.
- Portability: Code can be generated for different machines.



## Target Programming Language (TPL)

Our intermediate representation will be a generalised assembly code for a virtual three-address machine. We first present a subset of its syntax.

*Program*$_T$ → *Instruction*$^+$

*Instruction* → *StoreInstr* | *BinopInstr* | *UopInstr* | *JumpInstr* | *IOInstr*

It's called three-address machine because its instructions can take up to three arguments. For example:

```
ADDI $10,$12,R1
```



## Memory and registers

Values can be stored into (and read from) memory or registers:

- **Memory**: A sequence of cells (words) referenced (addressed) by their address or location. A memory reference is defined by:

  *Reference* → *$Location* | *Register*(*Offset*)
  *Location* ::= Integer

Memory locations start at address/location 0.

- **Registers**: Special fast access memory locations for efficient read and store operations. Machines have a limited number of registers.

  *Register* → **R1** | **R2** | … | **R***NumReg*

## The Store Instruction

*StoreInstr* → `STORE` *Arg*, *Res*

*Arg* → *IntegerLiteral* | *Register* | *Reference*

*Res* → *Register* | *Reference*

Stores the contents of *Arg* into *Res*, where:

*Examples:*

```
STORE 5, $7
STORE $1, R4
```



## Binary Operations

*BinopInstr* → $Op_I$ *Arg1, Arg2, Res*
*BinopInstr* → $Op_B$ *Arg1, Arg2, Res*
*BinopInstr* → $Op_C$ *Arg1, Arg2, Res*

$Op_I$ → `ADDI` | `SUBI` | `MULTI` | `DIVI`
$Op_B$ → `AND` | `OR` | `XOR`
$Op_C$ → `EQ` | `NE` | `GT` | `GE` | `LT` | `LE`

*BinopInstr* evaluates the binary operation taking *Arg1* and *Arg2* as arguments, and storing the result in *Res*. It performs an implicit STORE.



## Binary Operations

- $Op_I$: *Arg1* (*Arg2*), or the value stored in the location or register referenced by *Arg1* (*Arg2*), must be an integer. The result is an integer.
- $Op_B$: *Arg1* (*Arg2*), or the value stored in the location or register referenced by *Arg1* (*Arg2*), must be a boolean. The result is a boolean.
- Boolean values are 0 (false) and 1 (false)
- $Op_B$: *Arg1* (*Arg2*), or the value stored in the location or register referenced by *Arg1* (*Arg2*), must be an integer. The result is a boolean



## Examples Binary Operations

Assuming variables x,y and z are stored in locations 2,4 and 6 respectively:

- Variable assignment: z = x + y + 10

```
ADDI $2, $4, R1
ADDI R1, 10, R2
STORE R2, $6
```

- Test: “(z < 0) and (x == y)”

```
LT $6,0,R3
EQ $2, $4, R4
AND R3, R4, R5
```



## Unary and IO Operations

*UopInstr* → `UMINUS` *Arg*, *Res*
Negates the value denoted by *Arg* and stores the result in *Res*.

```
UMINUS R1, R2
```

*UopInstr* → `NOT` *Arg*, *Res*
Boolean negation. *Arg* must denote a boolean value.

```
STORE 0, R1
NOT R1, R2
```

*IOInstr* → `WRITEI` *Arg* | `READI` *Res*
Writes the value of *Arg* (integer) to standard output, and reads an integer from standard input into *Res*.



## Control Flow

*JumpInstr* → `LABEL` *LName*
Assigns label *LName* to next instruction. Labels usually have the same syntax as identifiers.

*JumpInstr* → `JMP` *LName*
Jumps to the instruction labeled by *LName*.

*JumpInstr* → `JMP0` *Arg, LName*
Jumps to the instruction labeled by *Lname if value of Arg is 0*.

*JumpInstr* → `JMP1` *Arg, LName*
Jumps to the instruction labeled by *Lname* if value of *Arg* is 1.

## Example: If Conditional

"if (x >= 0) then x = y + 1 else x = y + 2"

Assuming x and y are stored in locations 10 and 12, the translation of the code above may look like:

```
GE $10,0,R1
JMP0 R1,L1
ADDI $12,1,$10
JMP L2
LABEL L1
ADDI $12,2,$10
LABEL L2
```



## Example: While loop

"x = 0; while (x < 5) { x = x + 1; print(x); }"

An equivalent program in TPL:

```
STORE 0,$2
LABEL L1
LT $2,5,R1
JMP0 R1,L2
ADDI $2,1,$2
WRITEI $2
JMP L1
LABEL L2
```



## Goal: Code Generation

• We need to design an algorithm that automatically generates TPL code for any program written in our source programming language.

• We'll do this by code generation specification for each fragment (statements, expressions) of the source programming language grammar.

•We start next week.