07-Exercise-Instruction Selection

Compilers course

Masters in Informatics and Computing Engineering (MIEIC), 3rd Year

João M. P. Cardoso
Email: jmpc@fe.up.pt

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Instruction Selection, Slides 25-30

EXERCISE

Exercise

- Consider a microprocessor with the following instructions:
  - ADD rd = rs1 + rs2
  - ADDI rd = rs + c
  - SUB rd = rs1 − rs2
  - SUBI rd = rs − c
  - MUL rd = rs1* rs2
  - DIV rd = rs1/rs2
  - LOAD rd = M[rs + c]
  - STORE M[rs1 + c] = rs2
  - MOVEM M[rs1] = M[rs2]
- Where rd, rs identify registers of the architecture (from r0 to r31 and r0 stores the non-modified value 0) and c identifies literals

This exercise is based on the Andrew W. Appel "Tiger" book.

Exercise

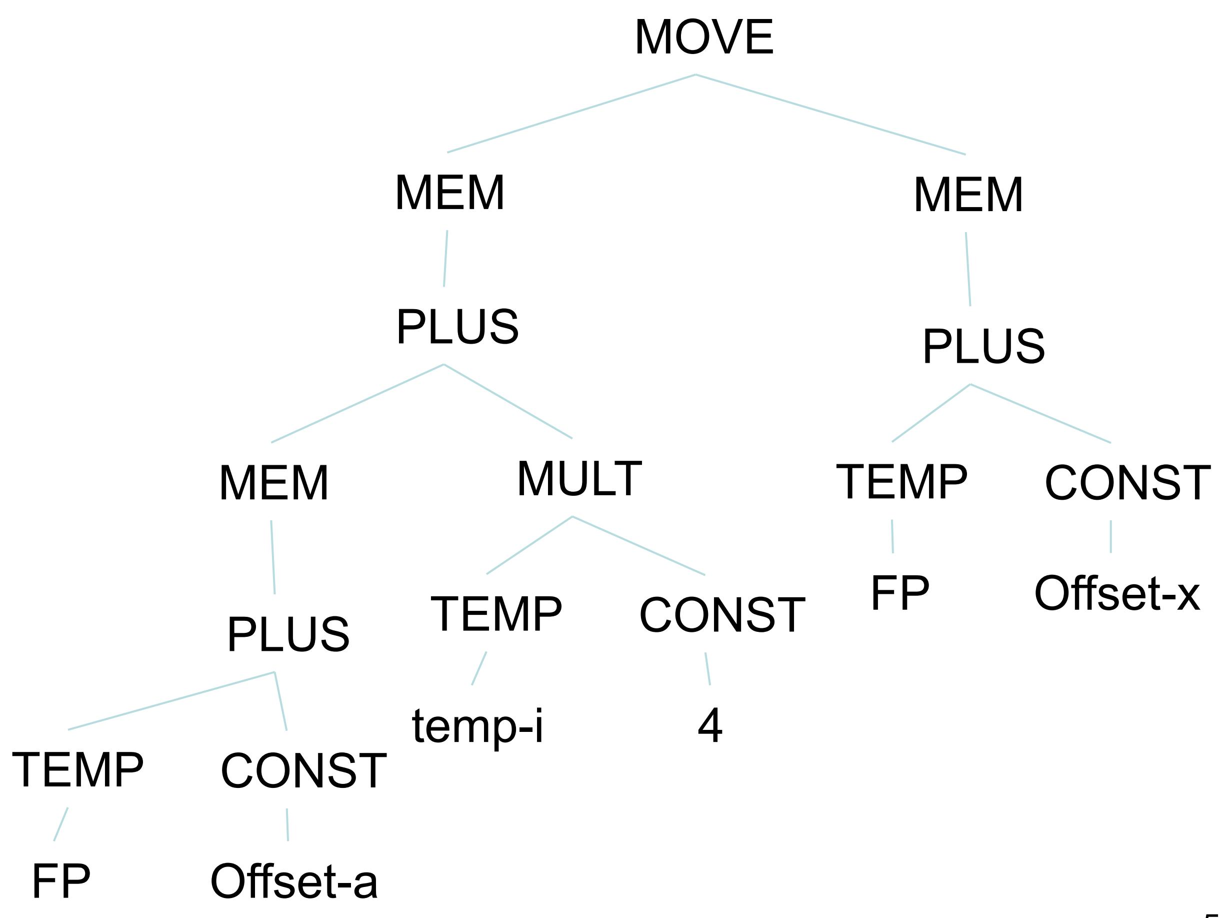
- The corresponding Instruction Tree Patterns are the following:

Instruction	Effect	IR Tree Pattern
—	r_i	TEMP r_i
add	$r_i \leftarrow r_j + r_k$	+
mul	$r_i \leftarrow r_j * r_k$	*
sub	$r_i \leftarrow r_j - r_k$	−
div	$r_i \leftarrow r_j / r_k$	/
addi	$r_i \leftarrow r_j + c$	+ CONST c + CONST c CONST c
subi	$r_i \leftarrow r_j - c$	− CONST c

Instruction	Effect	IR Tree Pattern
load	$r_i \leftarrow M[r_j + c]$	MEM + CONST c MEM + CONST c MEM CONST c MEM
store	$M[r_j + c] \leftarrow r_i$	MOVE MEM + CONST c MOVE MEM + CONST c MOVE MEM CONST c MOVE MEM
movem	$M[r_j] \leftarrow M[r_i]$	MOVE MEM MEM

Exercise

- Consider the input intermediate representation illustrated below for the statement: a[i] = x; (assuming *i* stored in a register identified by *r_i*, and *a* and x are frame residents), where *FP* represents the register with the frame pointer, *offset-a* and *offset*-x represent two constants, and *temp-i* identifies the variable *i*.



Exercise

a) Use individual node selection to generate the assembly instructions.
b) Use the Maximal-Munch algorithm for instruction selection and write the instructions generated.
c) Use dynamic programming to obtain an optimum solution for instruction selection (considering as goal the minimum number of instructions) and write the instructions generated.

Exercise

a) Use individual node selection to generate the assembly instructions.

ADDI r1 = r0 + offset_a

ADD r2 = r1 + FP

LOAD r3 = M[r2 + 0]

ADDI r4 = r0 + 4

MUL r5 = r4 * r_i

ADD r6 = r3 + r5

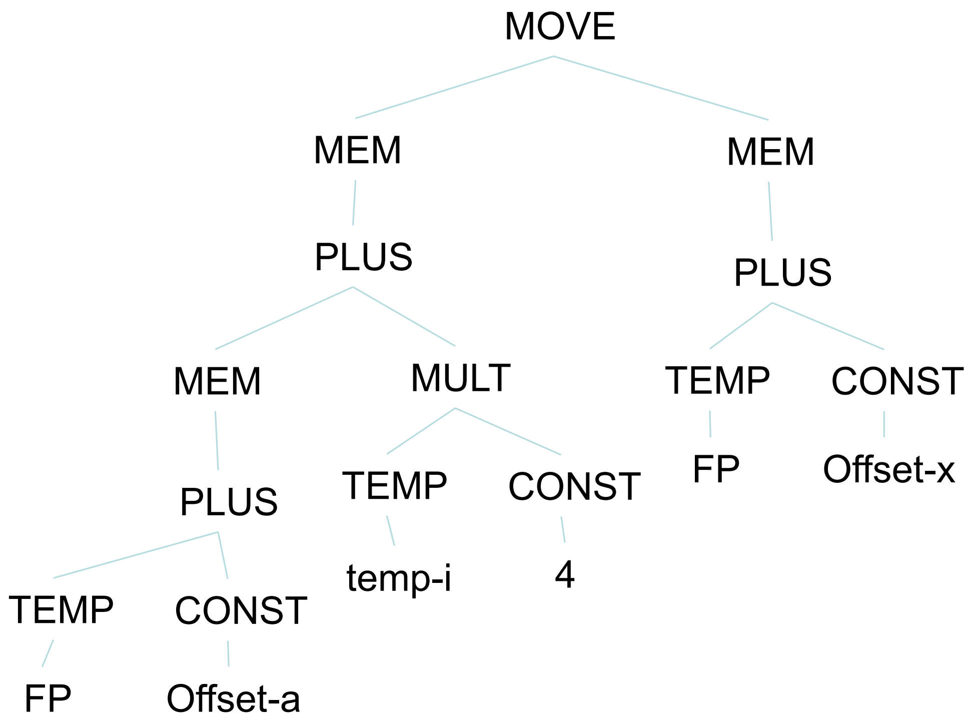
ADDI r7 = r0 + offset_x

ADD r8 = r7 + FP

LOAD r9 = M[r8 + 0]

STORE M[r6 + 0] = r9

9 registers, 10 instructions



Exercise

a) Use the Maximal-Munch algorithm for instruction selection and write the instructions generated.

LOAD r1 = M[FP + offset_a]

ADDI r2 = r0 + 4

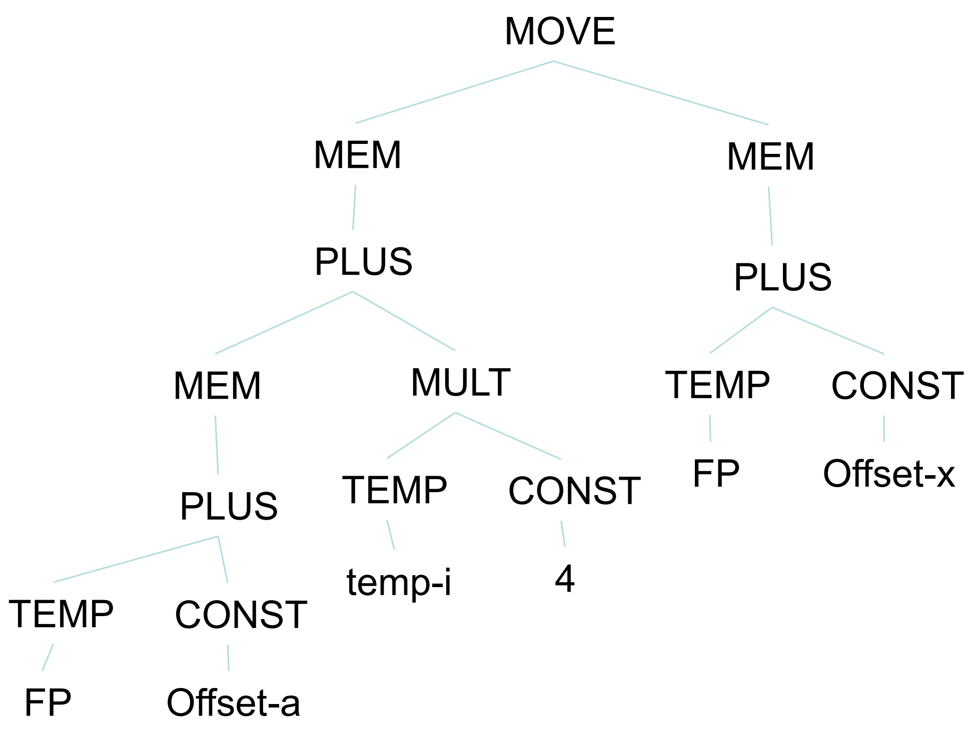
MUL r3 = r2 * r_i

ADD r4 = r1 + r3

ADD r5 = FP + offset_x

MOVEM M[r4] = M[r5]

5 registers, 6 instructions



Exercise

a) Use dynamic programming to obtain an optimum solution for instruction selection (considering as goal the minimum number of instructions) and write the instructions generated.

LOAD r1 = M[FP + offset_a]

ADDI r2 = r0 + 4

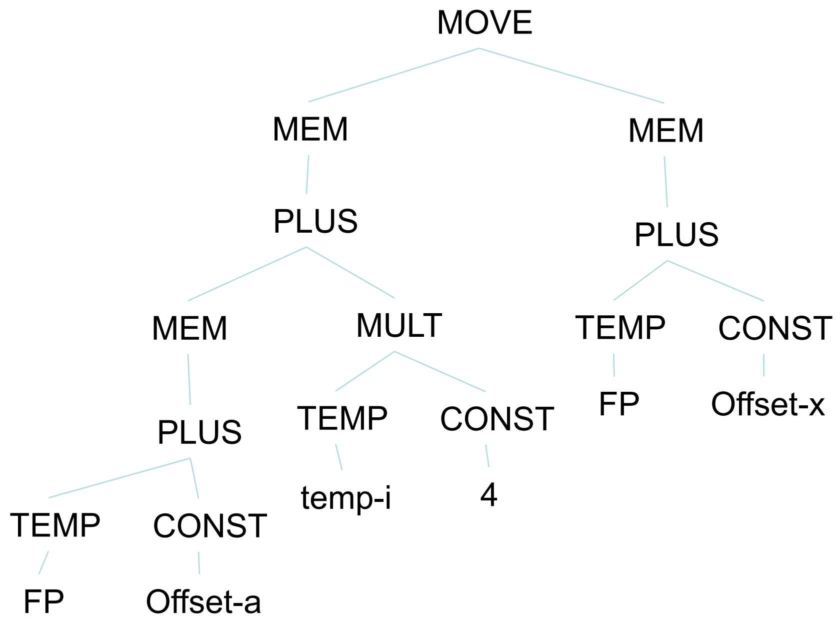
MUL r3 = r2 * r_i

ADD r4 = r1 + r3

LOAD r5 = M[FP + offset_x]

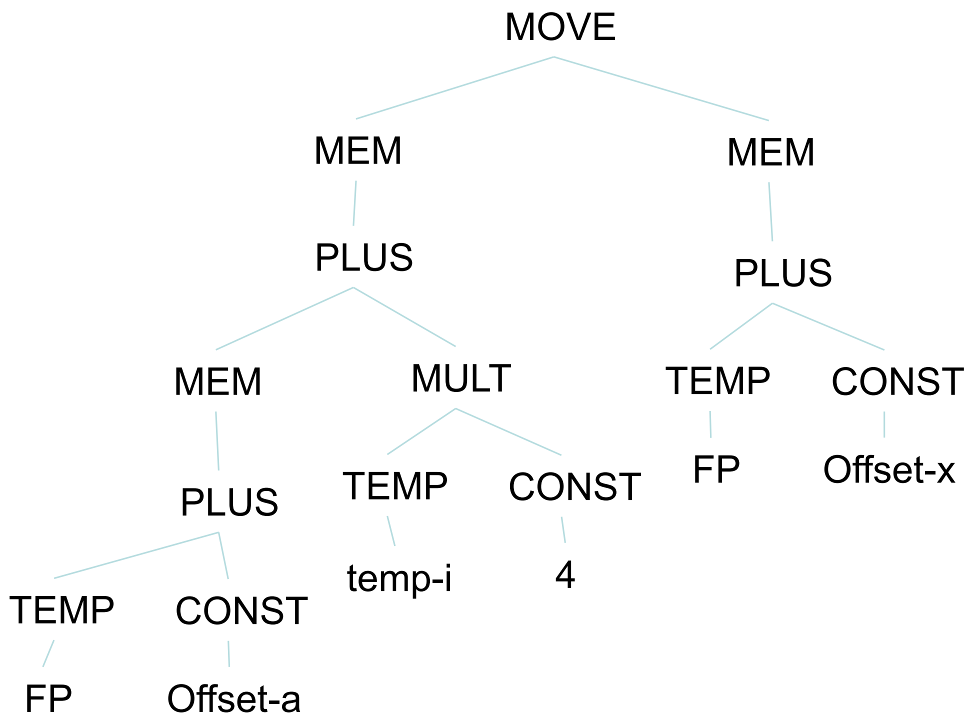
STORE M[r4] = r5

5 registers, 6 instructions



In this case, the optimal tree generated by Maximum Munch is also optimum...

- Consider the input intermediate representation illustrated below for the statement: a[i] = x; (assuming *i* stored in a register identified by *r_i*, and *a* and x are frame residents), where *FP* represents the register with the frame pointer, *offset-a* and *offset-x* represent two constants, and *temp-i* identifies the variable *i*.



Instruction	Effect	IR Tree Pattern
load	$r_i \leftarrow M[r_j + c]$	MEM(+(·, CONST c)); MEM(+(CONST c, ·)); MEM(CONST c); MEM(·)
store	$M[r_j + c] \leftarrow r_i$	MOVE(MEM(+(·, CONST c)), ·); MOVE(MEM(+(CONST c, ·)), ·); MOVE(MEM(CONST c), ·); MOVE(MEM(·), ·)
movem	$M[r_j] \leftarrow M[r_i]$	MOVE(MEM(·), MEM(·))

Instruction	Effect	IR Tree Pattern
—	r_i	TEMP r_i
add	$r_i \leftarrow r_j + r_k$	+(·, ·)
mul	$r_i \leftarrow r_j * r_k$	*(·, ·)
sub	$r_i \leftarrow r_j - r_k$	−(·, ·)
div	$r_i \leftarrow r_j / r_k$	/(·, ·)
addi	$r_i \leftarrow r_j + c$	+(·, CONST c); +(CONST c, ·); CONST c
subi	$r_i \leftarrow r_j - c$	−(·, CONST c)