1. (a) The Java programming language is compiled into an intermediate language called *Java byte code* which can be inspected using a *disassembler*. Somewhat simplified, it can be said that the disassembled representation contains a class definition which contains a list of methods. A method body is a list of two-operand or three-operand byte codes. An example of the output from the disassembler is shown below.

```
class HelloWorld {
  method SayHello () {
    two-operand-code exp1 exp2
    three-operand-code exp1 exp2 exp3
  }
}
```

Give a grammar for Java byte code as described above. Your grammar should be suitable for producing a parser which reads in the output of the Java byte code disassembler. Provide semantic actions for the construction of the abstract syntax representation of the input. You may assume the existence of any constructor functions which you need to use.

You should state your answer *either* in the form of the input to the Yacc/Bison parser generator (with the semantic actions coded in C) *or* in the form of the input to the CUP parser generator (with the semantic actions coded in Java) *or* in the form of the input to the ML/Yacc parser generator (with the semantic actions coded in Standard ML).

[15 marks]

(b) Is your grammar from part (a) LL(1)? If not, explain where it fails to be LL(1).

[5 marks]

(c) In writing a grammar for Java byte code as described above, one type of error which might arise is a *reduce/reduce* error. Briefly explain the cause of these errors and identify the part of the grammar which would have been most likely to cause a reduce/reduce error.

[5 marks]

2. (a) What is the purpose of the *semantic analysis* phase of the compilation process? Describe the input into this phase and the results which come out.

[4 marks]

(b) The Tiger compiler includes a `Symbol` module with functions which convert between strings and symbols. What advantages are gained from using symbols instead of strings in the Tiger compiler?

[6 marks]

(c) Describe the semantic analysis of the following two mutually recursive functions. The function `f` has two arguments of type `ty_a` and one of type `ty_b`. The function `g` has two arguments of type `ty_a` and a further two of type `ty_b`.

```
function f(a1, a2: ty_a, b: ty_b) : ty_b =
  if a1 = a2 then b else g(a1, a2, b, b)
function g (a1, a2: ty_a, b1, b2: ty_b) : ty_b =
  if b1 = b2 then f(a1, a1, b1) else f(a1, a2, b2)
```

[5 marks]

(d) The suggestion has been made that the Tiger language should include *static* variables, whose initialisations are performed only once. Values of static variables persist between calls to a function so it would be possible to implement a counter function as shown below.

```
function counter () : int =
let  static c : int := 0
in   (c := c + 1;  c)
end
```

(i) Is this extension to the Tiger language necessary or can the same function be implemented without the addition of static variables? If possible, show how the above function would be implemented.

[6 marks]

(ii) What difference would the inclusion of static variables make to the implementation of *environments* for the Tiger language?

[4 marks]

3. (a) The Jouette instruction set has six frequently-used arithmetic and memory instructions: ADD, SUB, ADDI, SUBI, LOAD and STORE. It has three less frequently-used instructions: MUL, DIV and MOVEM. What are the *tree patterns* which correspond to the six frequently-used instructions?

[6 marks]

(b) Consider the following Tiger language assignment.

```
x := a[i + 1]
```

Assume that the variable `i` is a register variable and both `x` and `a` are resident in the frame.

(i) Draw the intermediate representation tree which you would expect the Tiger compiler to generate for this assignment. Assume that the machine word size is 4 bytes.

[4 marks]

(ii) Show how your tree could be tiled when generating the Jouette instruction sequence which corresponds to this assignment.

[6 marks]

(iii) Give the Jouette instruction sequence which corresponds to your tiling.

[4 marks]

(c) What are the criteria to be used when selecting a tiling algorithm for use with a Complex Instruction Set Computer (CISC) architecture? How would these criteria differ for a Reduced Instruction Set Computer (RISC) architecture?

[5 marks]