THE UNIVERSITY OF AUCKLAND

First Semester, 2006

City Campus

Computer Science

Language Implementation

(Time allowed TWO hours)

FAMILY NAME:
PERSONAL NAMES:
STUDENT ID NUMBER:
LOGIN NAME:
SIGNATURE:

This Examination is out of 100 Marks. Attempt **ALL** questions. Write your answers in the spaces provided in this question and answer booklet. Do not remove the staples from the question and answer booklet. However, you may detach and remove the staples from the appendices.

1		10
2		23
3		15
4		7
5		14
6		16
7		15
Total		100

Continued ...

Print Student ID _____

1. JFlex [10 Marks]

Indicate at least 10 different kinds of errors in the following fragment of JFlex code. ("..." just means omitted code). Give a reason or appropriate correction for each one. Assume line breaks **are** syntactically important, but spaces **are not**. Assume comments **can** be nested.

```
%{
     int commentNest = 0;
     int lineCount = 0;
     ...
%}
%init{
     yybegin( NORMAL );
%init}
newline        =    [\r|\n|\r\n]
space          =    [\ \t]
ident          =    [A-Za-z0-9]*
number         =    [1-9]+
%state NORMAL, COMMENT
%%
<NORMAL> {
     if        { return token( sym.IF ); }
     else      { return token( sym.ELSE ); }
     (         { return token( sym.LEFT ); }
     )         { return token( sym.RIGHT ); }
     .         { return token( sym.DOT ); }
     ...
     "/*"      { commentNest++; return token( sym.COMMENT ); }
     "*/"      { --commentNest; yybegin( NORMAL ); }
     ident     { return token( sym.IDENT ); }
     number    { return token( sym.NUMBER ); }
     {space}   { }
     {newline} { lineCount++; }
     "."       { }
     }
<COMMENT> {
     "/*"      { commentNest++; }
     "*/"      { --commentNest; yybegin( NORMAL ); }
     {newline} { }
     .         { return token( sym.LEXERROR ); }
     }
<<EOF>>        { return token( sym.EOF ); }
```

(10 marks)

Continued ...

Print Student ID ____

2. Bottom Up LALR(1) Parsing [23 Marks]

Consider the CUP grammar in the **Appendix For Question 2**. Note that some rules are left recursive, while other rules are right recursive. Also note the rule for “ElementListOpt” that expands to empty.

(a) Using the information provided in the appendix, perform a shift-reduce LALR(1) parse of the input

```
second( [ X, Y | Z ], Y )
```

Assume “`second`” matches `NAME`; “`X`”, “`Y` ”, “`Z`”, match `VARIABLE`, and “`,`”, “`(`”, “`)`”, “`[`”, “`]`” and “`|`” match `COMMA`, `LEFT`, `RIGHT`, `LEFTSQ`, `RIGHTSQ` and `BAR`, respectively.

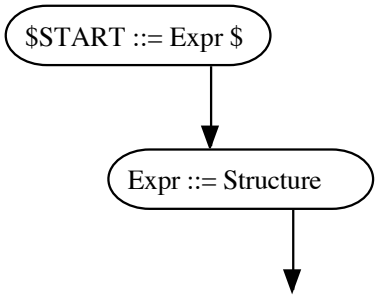
Stack									Token	Action	
$0									NME sec	Shift	
$0	NME 5								(	Shift	
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0											
$0	E 3	$ 22							$	Reduce	$St ::= E $
$0	$St -1									Accept	

(15 marks)

Continued ...

Print Student ID _____

(b) Draw the parse tree corresponding to the grammar rules used to parse this input

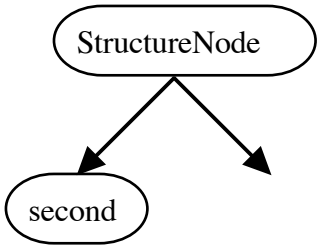


(5 marks)

Continued ...

Print Student ID _____

(c) Draw the abstract syntax tree built by the actions associated with the grammar rules.



(3 marks)

Continued ...

Print Student ID _____

3. Write a grammar for class declarations [15 Marks]

A class declaration is of the form

```
class classID
     extends superID
     implements ifaceID1, ifaceID2, ifaceID3 ... {
     MemberDecl1
     MemberDecl2
     ...
     }
```

where `classID`, `superID`, `ifaceID1`, `ifaceID2`, `ifaceID3` ... are all identifiers. The "`extends superID`" and "`implements ifaceID1, ifaceID2, ifaceID3 ...`" portions are optional.

A `MemberDecl` is either a `VarDecl` or `MethodDecl`.

A `VarDecl` is of the form

```
Type varID1, varID2, varID3 ... ;
```

where `varID1`, `varID2`, `varID3` ... are simple identifiers. There is no initialisation expression.

A `MethodDecl` is of the form

```
Type methodID ( VarDecl1 VarDecl2 ... ) {
     LocalDeclStmt1
     LocalDeclStmt2
     ...
     }
```

A `LocalDeclStmt` is either a `VarDecl` or `Statement`.

A `Type` can be a simple `typeID`, or an array type such as `[]typeID`, `[][]typeID`, etc., where `typeID` is an identifier.

For example, we could have the class declaration such as

```
class A extends B {
     []C x, y, z;
     D e( F p, q; [][]G r, s; ) {
          ...
          }
     }
```

Write a grammar for class declarations **in general**.

You do not have to define the grammar for a Statement.

You do not have to include any actions.

Continued ...

Print Student ID _____

Continued ...

Print Student ID _____

(15 marks)

Continued ...

Print Student ID _____

4. Write a BHP function [7 Marks]

Consider the BHP language of Assignment 2.

Write a BHP function to clone a compound object.

(7 marks)

Continued ...

Print Student ID _____

5. Show the run time stack for a B-- program [14 Marks]

Use the program written in the B-- language of assignment 3 in the **Appendix For Question 5**.

Complete the drawing of the data structure built for the global variables "`source`" and "`dest`".

Use multiple colours (other than red) for arrows, etc., to make your diagram clearer.

Display the stack frames (activation records) for all methods in the process of being invoked when the maximum level of nesting of method invocations occurs when the statement

```
reverse( source, &dest );
```

on line 53 is invoked, and the process is almost ready to return. At this stage the process should be executing the method "`transfer`" at line 35.

Indicate the appropriate values for each stack frame (activation record) you draw. The line numbers on the left-hand side of the program should be used to represent the return address. Draw appropriate arrows for the var parameters, and pointers to objects. For var parameters, make sure you indicate the exact variable pointed to in an object very clearly. Represent `List` nodes as shown in the sample entries.

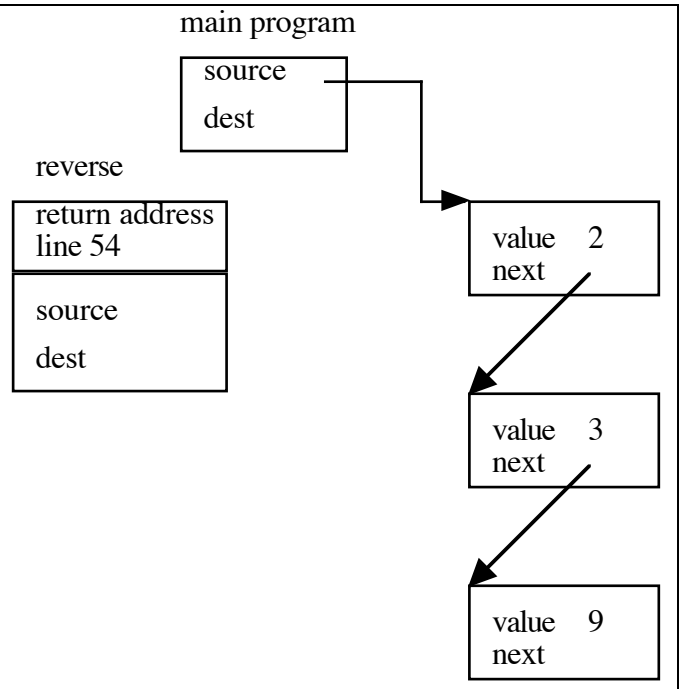
Also indicate the output generated by the complete execution of the program.

Output generated:

(2 marks)

Continued ...

Print Student ID _____



(12 marks)

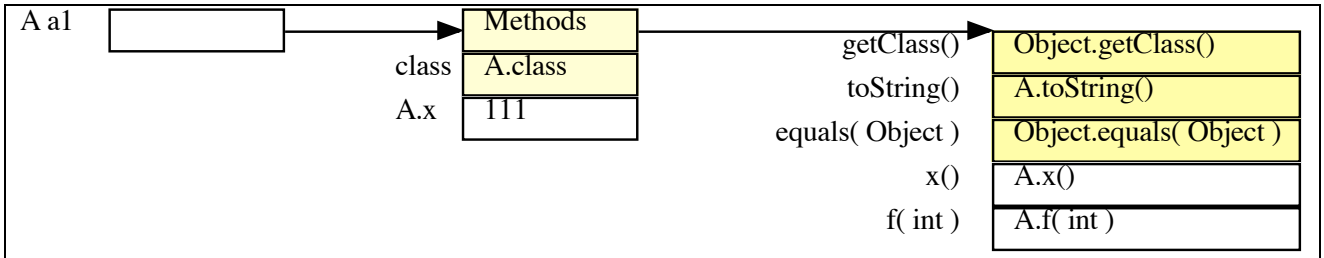
Continued ...

Print Student ID _____

6. Implementation of object oriented languages [16 Marks]

Use the Java program in the **Appendix For Question 6**.

(a) Draw a diagram showing the data structures (variable, field table, method table, etc) created for the variables `a1`, `a2`, `b`, `c`, within the method `Main.main`. Shared data structures should be drawn only once.



(11 marks)

Continued ...

Print Student ID _____

(b) Indicate the output generated by the method Main.main.

```
Constructor A









a1.getClass() = class A

a2.getClass() =

b.getClass() =

c.getClass() =

a1 =

a2 =

b =

c =

a1.x = 111

a2.x =

b.x =

c.x =

a1.x() = 111

a2.x() =

b.x() =

c.x() =

a1.f( 'a' ) =

a2.f( 'a' ) =

b.f( 'a' ) =

c.f( 'a' ) =
```

(5 marks)

Print Student ID _____

7. Code generation [15 Marks]

Consider the program written in the B-- language of assignment 3, and its partial translation into Alpha assembly language in the **Appendix For Question 7.**

Fill in the missing code to complete appropriate portions of the generated Alpha assembly language.

An appendix is also provided describing common Alpha instructions.

```
    public a_10:
                ... Code to allocate space for Variable "a"
                                                                    (1 mark)
        align;
...
        public block methodTable {
            const {
            public enter:
                ... Code for method table for "A"
                                                                    (1 mark)
            } const
        } block methodTable
...
            code {
                align;
            public enter:
                lda $sp, -max($sp);
                stq $ra, savRet($sp);
                stq $fp, savFP($sp);
                stq $ip, savIP($sp);
                mov $nip, $ip;
                mov $sp, $fp;
                {
                    ... Code to perform initialisation of any instance of "A"
                                                                    (4 marks)
                }
            return:
                mov $ip, $nip;
                ldq $ip, savIP($sp);
                ldq $fp, savFP($sp);
                ldq $ra, savRet($sp);
                lda $sp, +max($sp);
                ret;
            } code
```

Continued ...

Print Student ID _____

```
// Initialisation code for globals
public block global {
    public block init extends proc.sav0 uses proc {
        code {
        public enter:
            ...
            {
```

```
                ... Code to initialise "a"
```

(4 marks)

```
            }
            {
```

```
                ... Code to perform "a.set( 5 );"
```

(5 marks)

```
            }
            ...
            ret;
        } code
       ...
    } block init
} block global
```

_____End of Questions_____

Print Student ID _____

This Page is left blank for questions that overflow

Continued ...

Print Student ID _____

This Page is left blank for questions that overflow

Continued ...

Print Student ID _____

This Page is left blank for questions that overflow

Appendix For Question 2

Grammar

```
terminal
     COMMA, LEFT, RIGHT, LEFTSQ, RIGHTSQ, BAR;

terminal String INTVALUE;
terminal String NAME;
terminal String VARIABLE;

nonterminal ExprListNode
     ExprList;
nonterminal ExprNode
     Expr, Structure, List, ElementListOpt, ElementList;

start with Expr;

ExprList::=
          Expr:expr
          {:
          RESULT = new ExprListNode( expr );
          :}
     |
          ExprList:exprList COMMA Expr:expr
          {:
          exprList.addElement( expr );
          RESULT = exprList;
          :}
     ;

Expr::=
          INTVALUE:value
          {:
          RESULT = new IntValueNode( new Integer( value ).intValue() );
          :}
     |
          NAME:name
          {:
          RESULT = new NameNode( name );
          :}
     |
          VARIABLE:name
          {:
          RESULT = new VariableNode( name );
          :}
     |
          Structure:structure
          {:
          RESULT = structure;
          :}
     |
          List:list
          {:
          RESULT = list;
          :}
     ;
```

Appendix For Question 2 Continued On Next Page ....

Appendix For Question 2 Continued ...

```
Structure::=
          NAME:name LEFT ExprList:exprList RIGHT
          {:
          RESULT = new StructureNode( name, exprList );
          :}
     ;

List::=
          LEFTSQ ElementListOpt:elementList RIGHTSQ
          {:
          RESULT = elementList;
          :}
     ;

ElementListOpt::=
          ElementList:elementList
          {:
          RESULT = elementList;
          :}
     |
          /* Empty */
          {:
          RESULT = new EmptyListNode();
          :}
     ;

ElementList::=
          Expr:expr
          {:
          RESULT = new NonEmptyListNode( expr, new EmptyListNode() );
          :}
     |
          Expr:expr COMMA ElementList:elementList
          {:
          RESULT = new NonEmptyListNode( expr, elementList );
          :}
     |
          Expr:expr BAR Expr:tail
          {:
          RESULT = new NonEmptyListNode( expr, tail );
          :}
     ;
```

Appendix For Question 2 Continued On Next Page ....

Appendix For Question 2 Continued ...

Rules

```
[0] $START ::= Expr EOF
[1] ExprList ::= Expr
[2] ExprList ::= ExprList COMMA Expr
[3] Expr ::= INTVALUE
[4] Expr ::= NAME
[5] Expr ::= VARIABLE
[6] Expr ::= Structure
[7] Expr ::= List
[8] Structure ::= NAME LEFT ExprList RIGHT
[9] List ::= LEFTSQ ElementListOpt RIGHTSQ
[10] ElementListOpt ::= ElementList
[11] ElementListOpt ::=
[12] ElementList ::= Expr
[13] ElementList ::= Expr COMMA ElementList
[14] ElementList ::= Expr BAR Expr
```

Action Table

```
From state #0
     LEFTSQ:SHIFT(state 6) INTVALUE:SHIFT(state 4) NAME:SHIFT(state 5)
     VARIABLE:SHIFT(state 7)
From state #1
     EOF:REDUCE(rule 6) COMMA:REDUCE(rule 6) RIGHT:REDUCE(rule 6)
     RIGHTSQ:REDUCE(rule 6) BAR:REDUCE(rule 6)
From state #2
     EOF:REDUCE(rule 7) COMMA:REDUCE(rule 7) RIGHT:REDUCE(rule 7)
     RIGHTSQ:REDUCE(rule 7) BAR:REDUCE(rule 7)
From state #3
     EOF:SHIFT(state 22)
From state #4
     EOF:REDUCE(rule 3) COMMA:REDUCE(rule 3) RIGHT:REDUCE(rule 3)
     RIGHTSQ:REDUCE(rule 3) BAR:REDUCE(rule 3)
From state #5
     EOF:REDUCE(rule 4) COMMA:REDUCE(rule 4) LEFT:SHIFT(state 16)
     RIGHT:REDUCE(rule 4) RIGHTSQ:REDUCE(rule 4) BAR:REDUCE(rule 4)
From state #6
     LEFTSQ:SHIFT(state 6) RIGHTSQ:REDUCE(rule 11) INTVALUE:SHIFT(state 4)
     NAME:SHIFT(state 5) VARIABLE:SHIFT(state 7)
From state #7
     EOF:REDUCE(rule 5) COMMA:REDUCE(rule 5) RIGHT:REDUCE(rule 5)
     RIGHTSQ:REDUCE(rule 5) BAR:REDUCE(rule 5)
From state #8
     RIGHTSQ:SHIFT(state 15)
From state #9
     COMMA:SHIFT(state 11) RIGHTSQ:REDUCE(rule 12) BAR:SHIFT(state 12)
From state #10
     RIGHTSQ:REDUCE(rule 10)
From state #11
     LEFTSQ:SHIFT(state 6) INTVALUE:SHIFT(state 4) NAME:SHIFT(state 5)
     VARIABLE:SHIFT(state 7)
From state #12
     LEFTSQ:SHIFT(state 6) INTVALUE:SHIFT(state 4) NAME:SHIFT(state 5)
     VARIABLE:SHIFT(state 7)
```

Appendix For Question 2 Continued On Next Page ....

Appendix For Question 2 Continued ...

```
From state #13
     RIGHTSQ:REDUCE(rule 14)
From state #14
     RIGHTSQ:REDUCE(rule 13)
From state #15
     EOF:REDUCE(rule 9) COMMA:REDUCE(rule 9) RIGHT:REDUCE(rule 9)
     RIGHTSQ:REDUCE(rule 9) BAR:REDUCE(rule 9)
From state #16
     LEFTSQ:SHIFT(state 6) INTVALUE:SHIFT(state 4) NAME:SHIFT(state 5)
     VARIABLE:SHIFT(state 7)
From state #17
     COMMA:SHIFT(state 19) RIGHT:SHIFT(state 20)
From state #18
     COMMA:REDUCE(rule 1) RIGHT:REDUCE(rule 1)
From state #19
     LEFTSQ:SHIFT(state 6) INTVALUE:SHIFT(state 4) NAME:SHIFT(state 5)
     VARIABLE:SHIFT(state 7)
From state #20
     EOF:REDUCE(rule 8) COMMA:REDUCE(rule 8) RIGHT:REDUCE(rule 8)
     RIGHTSQ:REDUCE(rule 8) BAR:REDUCE(rule 8)
From state #21
     COMMA:REDUCE(rule 2) RIGHT:REDUCE(rule 2)
From state #22
     EOF:REDUCE(rule 0)
```

Appendix For Question 2 Continued On Next Page ....

Appendix For Question 2 Continued ...

Reduce (Go To) Table

```
From state #0:
     Expr:GOTO(3)
     Structure:GOTO(1)
     List:GOTO(2)
From state #1:
From state #2:
From state #3:
From state #4:
From state #5:
From state #6:
     Expr:GOTO(9)
     Structure:GOTO(1)
     List:GOTO(2)
     ElementListOpt:GOTO(8)
     ElementList:GOTO(10)
From state #7:
From state #8:
From state #9:
From state #10:
From state #11:
     Expr:GOTO(9)
     Structure:GOTO(1)
     List:GOTO(2)
     ElementList:GOTO(14)
From state #12:
     Expr:GOTO(13)
     Structure:GOTO(1)
     List:GOTO(2)
From state #13:
From state #14:
From state #15:
From state #16:
     ExprList:GOTO(17)
     Expr:GOTO(18)
     Structure:GOTO(1)
     List:GOTO(2)
From state #17:
From state #18:
From state #19:
     Expr:GOTO(21)
     Structure:GOTO(1)
     List:GOTO(2)
From state #20:
From state #21:
From state #22:
```

Appendix For Question 5

```
1 class List
2  begin
3      int value;
4      ^List next;
5  end
6
7 int freeNode = 0;
8 [ 20 ]List nodeHeap;
9
10 ^List new( int value; ^List next; )
11     begin
12          ^List node = nodeHeap[ freeNode++ ];
13          node.value = value;
14          node.next = next;
15          return node;
16     end
17
18 void printList( ^List a; )
19     begin
20          printf( "{ " );
21          while a != null do
22               printf( "%d", a.value );
23               a = a.next;
24               if a != null then
25                    printf( ", " );
26               end
27          end
28          printf( " }\n" );
29     end
30
31 void transfer( int level; ^List source1, source2; ^^List dest; )
32     begin
33          if source1 == null then
34               dest^ = source2;
35               // Show state at this point
36               else
37               transfer( level + 1, source1.next,
38                    new( source1.value, source2 ), dest );
39          end
40     end
41
42 void reverse( ^List source; ^^List dest; )
43     begin
44          transfer( 0, source, null, dest );
45     end
46
47 ^List source =
48     new( 2,
49     new( 3,
50     new( 9,
51          null ) ) );
52 ^List dest;
53 reverse( source, &dest );  // Inside this invocation
54 printList( source );
55 printList( dest );
56
```

Appendices Continued ...

Appendix For Question 6

```
class A {
     public int x = 111;
     public A() { System.out.println( "Constructor A" ); }
     public int x() { return x; }
     public String toString() { return "A"; }
     public String f( int z ) { return "A.f( " + z + " )"; }
     }

class B extends A {
     public int x = 222;
     public B() { System.out.println( "Constructor B" ); }
     public String f( char z ) { return "B.f( \'" + z + "\' )"; }
     public String f( int z ) { return "B.f( " + z + " )"; }
     }

class C extends B {
     public int x = 333;
     public C() { System.out.println( "Constructor C" ); }
     public String toString() { return "C"; }
     public int x() { return x; }
     }

public class Main {
     public final static void main( String[] arg ) {
          A a1 = new A();
          A a2 = new B();
          C c = new C();
          B b = c;

          System.out.println( "a1.getClass() = " + a1.getClass() );
          System.out.println( "a2.getClass() = " + a2.getClass() );
          System.out.println( "b.getClass() = " + b.getClass() );
          System.out.println( "c.getClass() = " + c.getClass() );

          System.out.println( "a1 = " + a1 );
          System.out.println( "a2 = " + a2 );
          System.out.println( "b = " + b );
          System.out.println( "c = " + c );

          System.out.println( "a1.x = " + a1.x );
          System.out.println( "a2.x = " + a2.x );
          System.out.println( "b.x = " + b.x );
          System.out.println( "c.x = " + c.x );

          System.out.println( "a1.x() = " + a1.x() );
          System.out.println( "a2.x() = " + a2.x() );
          System.out.println( "b.x() = " + b.x() );
          System.out.println( "c.x() = " + c.x() );

          // 'a' is ASCII 97
          System.out.println( "a1.f( \'" + 'a' + "\' ) = " + a1.f( 'a' ) );
          System.out.println( "a2.f( \'" + 'a' + "\' ) = " + a2.f( 'a' ) );
          System.out.println( "b.f( \'" + 'a' + "\' ) = " + b.f( 'a' ) );
          System.out.println( "c.f( \'" + 'a' + "\' ) = " + c.f( 'a' ) );
          }
     }
```

Appendices Continued ...

Appendix For Question 7

B-- Program

```
class A
     begin
          int x = 3;
          int y = 4;
          int get()
               begin
                    return x;
               end
          void set( int x; )
               begin
                    this.x = x;
               end
     end

A a;

a.set( 5 );
```

Translation of B-- Program into Alpha Assembly Language

```
...
public block main uses register {
    // Global variable declarations for main
    data {
    public true_0:
        quad;
        align;
    public false_8:
        quad;
        align;
    public a_10:
                ... Code to allocate space for Variable "a"
        align;
    } data
    public block A_0 {
        // Field table layout for main.A_0
        public block field {
            local {
            public methodTablePtr:
                quad;
            public x_8:
                quad;
                align;
            public y_10:
                quad;
                align;
            public max:
            } local
        } block field
```

Appendix For Question 7 Continued On Next Page ....

Appendix For Question 7 Continued

```
// Method table layout for main.A_0
public block method {
    local {
    public get_0:
        quad;
        align;
    public set_8:
        quad;
        align;
    public max:
    } local
} block method
// Actual method table for main.A_0
public block methodTable {
    const {
    public enter:
        ... Code for method table for "A"
    } const
} block methodTable
// Code for methods for main.A_0
public block methodCode {
    public block get_0 extends proc.sav0 uses proc {
        code {
            align;
        public enter:
            ... Code for the body of "get()"
        } code
        local {
        public locals:
        public params:
        public max:
        } local
    } block get_0
    public block set_8 extends proc.sav0 uses proc {
        code {
            align;
        public enter:
            ... Code for the body of "set( int x; )"
        } code
        local {
        public locals:
        public params:
        public x_0:
            quad;
            align;
        public max:
        } local
    } block set_8
} block methodCode
```

Appendix For Question 7 Continued On Next Page ....

Appendix For Question 7 Continued

```
        // Code to initialise an instance of class A_0
         public block initInstance extends proc.sav0 uses proc {
             code {
                 align;
             public enter:
                 lda $sp, -max($sp);
                 stq $ra, savRet($sp);
                 stq $fp, savFP($sp);
                 stq $ip, savIP($sp);
                 mov $nip, $ip;
                 mov $sp, $fp;
                 {
                     ... Code to perform initialisation of any instance of "A"
                 }
             return:
                 mov $ip, $nip;
                 ldq $ip, savIP($sp);
                 ldq $fp, savFP($sp);
                 ldq $ra, savRet($sp);
                 lda $sp, +max($sp);
                 ret;
             } code
             local {
             public max:
             } local
         } block initInstance
     } block A_0
     // Code for methods for main
     public block methodCode {
         ...
     } block methodCode
     // Initialisation code for globals
     public block global {
         public block init extends proc.sav0 uses proc {
             code {
             public enter:
                 ...
                 {
                     ... Code to initialise "a"
                 }
                 {
                     ... Code to perform "a.set( 5 );"
                 }
                 ...
                 ret;
             } code
            ...
         } block init
     } block global
     code {
     public enter:
         clr $ip;
         clr $nip;
         bsr global.init.enter;
         clr $a0;
         bsr Sys.exit.enter;
     } code
 } block main
```

Appendices Continued ...

Commonly used Alpha instructions

Integer operate instructions

```
Opcode $regA, $regB, $regC
```

intReg[regC] = intReg[regA] op intReg[regB]

```
Opcode $regA, constantB, $regC
```

The constant is an 8 bit unsigned constant.
intReg[regC] = intReg[regA] op constantB

Arithmetic integer operate instructions

addq	add	+
subq	subtract	-
mulq	multiply	*
divq/divqu	divide, signed/unsigned	/
modq/modqu	modulo, signed/unsigned	%
s8addq	scaled 8 add	8*operandA+operandB

Shift integer operate instructions

sll	shift left logical	<<
srl	shift right logical	>>>
sra	shift right arithmetic	>>

Compare integer operate instructions

cmpeq	compare equal	==
cmplt/cmpult	compare less than signed/unsigned	<
cmple/cmpule	compare less than or equal signed/unsigned	<=

Logical integer operate instructions

and	and	&
bic	bit clear	& ~
bis/or	bit set/or	|
eqv/xornot	equivalent/exclusive or not	^ ~
ornot	or not	| ~
xor	exclusive or	^

Appendices Continued ...

Memory instructions

```
Opcode $regA, displacement($regB)
Opcode $regA, ($regB)
Opcode $regA, constant
```

The displacement or constant is a 16 bit signed constant.

Load address instruction

intReg[regA] = displacement + intReg[regB]

lda	load address

Load memory instructions

intReg[regA] = Memory[displacement + intReg[regB]]

ldq	load quadword
ldl	load longword
ldbu	load byte unsigned

Store memory instructions

Memory[displacement + intReg[regB]] = intReg[regA]

stq	store quadword
stl	store longword
stb	store byte

Branch instructions

Conditional branch instructions

```
Opcode $regA, destination
```

if (condition holds for intReg[regA])
    programCounter = destination

beq	branch equal
bne	branch not equal
blt	branch less than
ble	branch less than or equal
bgt	branch greater than
bge	branch greater than or equal
blbs	branch low bit set
blbc	branch low bit clear

Unconditional branch instructions

```
Opcode destination;
```

programCounter = destination // br

intReg[ra] = programCounter // bsr
programCounter = destination

br	branch
bsr	branch to subroutime

Appendices Continued ...

Jump instruction

```
Opcode ($regA);
```

programCounter = intReg[regA] // jmp

intReg[ra] = programCounter // jsr
programCounter = intReg[regA]

jmp	jump
jsr	jump to subroutine

Return instruction

programCounter = intReg[ra]

ret	return

Callpal instruction

```
call_pal constant;
```

The constant is a 26 bit constant.

call_pal	call PALcode

Pseudoinstructions

Load immediate

```
ldiq $regA, constant
```

The constant is a 64 bit constant.
intReg[regA] = constant

ldiq	load immediate quadword

Clear

```
clr $regA
```

intReg[regA] = 0

clr	clear

Unary pseudoinstructions

```
Opcode $regB, $regC
```

intReg[regC] = op intReg[regB]

```
Opcode constantB, $regC
```

The constant is an 8 bit unsigned constant.
intReg[regC] = op constantB

mov	move
negq	negate

_____End of Appendices_____