6. **Constructing Static Single Assignment (SSA) form (20 p)**

Given the following program fragment:

```
s := 0;
i := 0;
while (i<100){
      s := s + a[i];
      i := i + 1;
}
print(s);
```

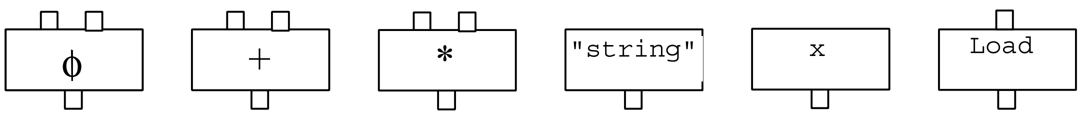
where `s, i` are local integer variables and `a` is an integer array with `100` elements.

(a) Construct a Basic Block Graph for this fragment. Fill in sequential code into the basic blocks. Use may use:

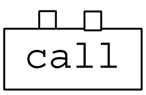
1. Read access from and assignments to local variables (e.g. `s :=i` assigns the content of the local variable `i` to the local variable `s`),
2. Address constants (assume `a0` is the constant address of `a[0]`) and address arithmetic (assume an address addresses a Byte and an Integer requires 4 bytes).
3. Load operation to get the Integer content of an address (e.g. `Load a0` gets the content of `a0`, i.e. it gets `a[0]`),
4. Integer constants and Integer arithmetic,
5. String constants,
6. Integer comparison (<), and
7. Procedure calls (e.g. `call("print", s)` calls procedure `print` on actual parameter `s`).

(b) Construct an SSA representation of this fragment's Basic Block Graph. Use indices for the different versions of the local variables. Show both the intermediate situation with immature ϕ' node and the final situation.

(c) Construct the SSA graph with local variables displayed as edges. Name edges after the corresponding local variable. Use the following nodes for operations:

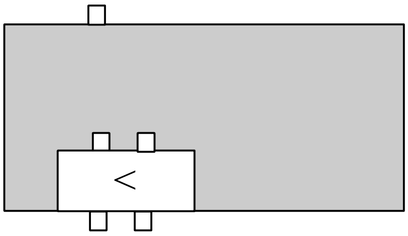


φ nodes | int/address plus | int/address multiplicat. | string constant | int/address constant `x` | Load addr. content

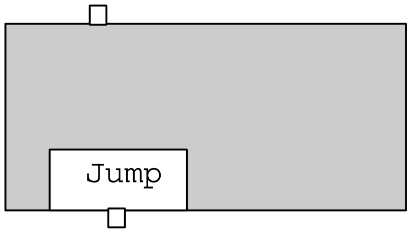


call nodes (taking the name of the procedure to call and the actual parameter value).

Blocks ending in a conditional jump and an unconditional jump, resp., are denoted by:



resp.



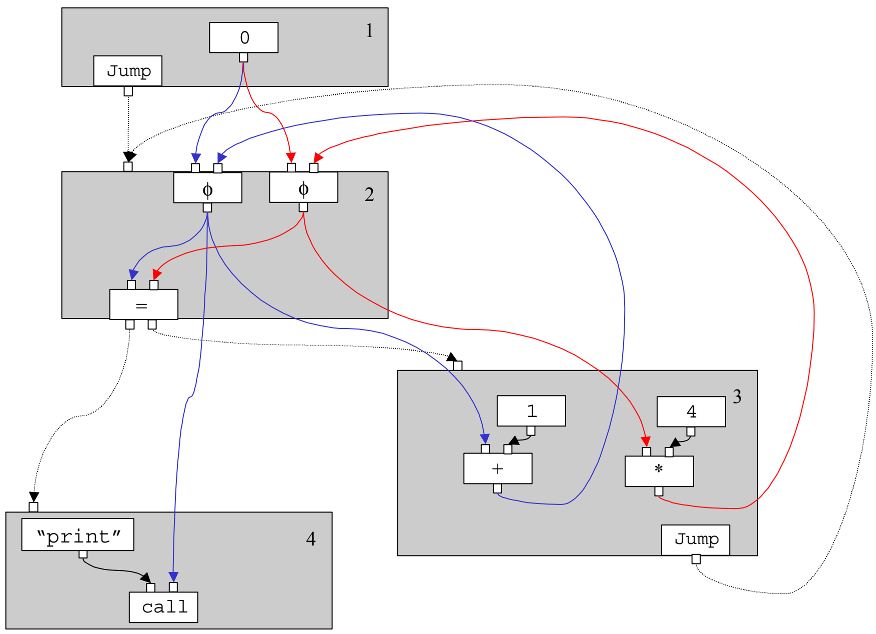
Hints: Block entries and exits are connected by control edges (use dashed lines). For blocks ending in a conditional jump, the *left* exit is the `false`, the *right* is the `true` exit. Operation entries and exits are connected by data edges (use solid lines). Ignore memory edges.

(d) Deconstruct the SSA graph.

1. Introduce variables for edges,
2. Remove ϕ nodes,
3. Determine live variables,
4. Compute the register interference graph
5. Compute a register allocation by graph coloring.

7. **Data Flow Analyses on SSA form (10 p)**

Given the following SSA graph:



(a) Reconstruct a program fragment represented by the graph. What is the value of the actual argument of the call to `"print"`?

Hint: Choose local variable names arbitrarily. Mind the conditional jump that terminates block 2 jumps to the *left* (the `false` exit) on inequality and to the *right* (the `true` exit) on equality.

(b) Perform context-**in**sensitive data flow analysis. What is the analyzed value of the actual argument of the call to "`print`"?

Hint: Assume the following definitions:

- Abstract Integer values: { ⊥, 0, 1, 2, …, maxint, ⊤}
- Context-insensitive transfer functions T_+, T_*:

  $T_{+,*}(\bot, x) = T_{+,*}(x, \bot) = \bot$

  $T_{+,*}(\top, x) = T_{+,*}(x, \top) = \top$

  For $a,b \in$ Integer:

  $T_+(a,b) = a+b$ (usual Integer addition)

  $T_*(a,b) = a*b$ (usual Integer multiplication)
- Context-insensitive meet function:

  $T\phi(\bot, x) = T\phi(x, \bot) = x$

  $T\phi(\top, x) = T\phi(x, \top) = \top$

  $T\phi(x, x) = x$

  $T\phi(x, y) = \top$

(c) Perform context-sensitive data flow analysis. What is the analyzed value of the actual argument of the call to "`print`"? What is missing in the analysis for deriving the actually expected result as in answer to 7 (a)?

Hint: Use the generalization of the data flow values to χ terms and the generalization of the context-insensitive transfer functions context-sensitive transfer functions.