

# **Exam 3**

## **601.428/628 Compilers and Interpreters**

December 21, 2021

Complete all questions.

Closed book, no use of electronic resources is permitted.

You may use two sheets of hand-written notes.

Time: 90 minutes.

I affirm that I have completed this exam without unauthorized assistance from any person, materials, or device.

Signed: \_\_\_\_\_

Print name: \_\_\_\_\_

Date: \_\_\_\_\_

# Reference

Reference for high-level instructions:

Instruction		Meaning
ldci	<i>Vrd</i> , \$N	Store integer constant \$N to <i>Vrd</i>
addi	<i>Vrd</i> , <i>Op</i> , <i>Op</i>	Store sum of operands to <i>Vrd</i>
subi	<i>Vrd</i> , <i>Op</i> , <i>Op</i>	Store difference of operands (left - right) to <i>Vrd</i>
muli	<i>Vrd</i> , <i>Op</i> , <i>Op</i>	Store product of operands to <i>Vrd</i>
divi	<i>Vrd</i> , <i>Op</i> , <i>Op</i>	Store result of dividing operands (left / right) to <i>Vrd</i>
modi	<i>Vrd</i> , <i>Op</i> , <i>Op</i>	Store remainder after dividing operands (left / right) to <i>Vrd</i>
negi	<i>Vrd</i> , <i>Op</i>	Store negation of operand to <i>Vrd</i>
localaddr	<i>Vrd</i> , \$N	Store address of memory location at offset of \$N bytes into local memory storage area to <i>Vrd</i>
ldi	<i>Vrd</i> , ( <i>Vra</i> )	Store value loaded from memory location ( <i>Vra</i> ) to <i>Vrd</i>
sti	( <i>Vra</i> ), <i>Vrb</i>	Store value in <i>Vrb</i> to memory location ( <i>Vra</i> )
readi	<i>Vrd</i>	Read integer input value and store it to <i>Vrd</i>
writei	<i>Vra</i>	Write integer value in <i>Vra</i> to output
jmp	<i>label</i>	Unconditional jump to <i>label</i>
cmpi	<i>Op</i> , <i>Op</i>	Compare operands
je	<i>label</i>	Conditional jump to <i>label</i> if in previous comparison, left = right
jne	<i>label</i>	Conditional jump to <i>label</i> if in previous comparison, left ≠ right
jlt	<i>label</i>	Conditional jump to <i>label</i> if in previous comparison, left < right
jlte	<i>label</i>	Conditional jump to <i>label</i> if in previous comparison, left ≤ right
jgt	<i>label</i>	Conditional jump to <i>label</i> if in previous comparison, left > right
jgte	<i>label</i>	Conditional jump to <i>label</i> if in previous comparison, left ≥ right
mov	<i>Vrd</i> , <i>Vra</i>	Store value in <i>Vra</i> to <i>Vrd</i>

Notes:

- Virtual registers are *vr0*, *vr1*, etc.
- *Vrd* means a destination virtual register (modified by the instruction)
- *Vra* and *Vrb* are source virtual registers (not modified by the instruction)
- (*Vra*) and (*Vrb*) mean a memory reference using a virtual register as a pointer, e.g., (*vr0*)
- \$N means an integer constant (e.g., \$42)
- *Op* means a source operand (either source virtual register or integer constant)
- *label* means a target label
- All values are 64-bit signed integers

**Question 1.** [25 points] Consider the following high-level basic block:

```
1:  localaddr vr13, $0
2:  ldci      vr14, $10
3:  muli      vr15, vr0, vr14
4:  addi      vr16, vr15, vr2
5:  muli      vr17, vr16, $8
6:  addi      vr18, vr13, vr17
7:  ldi       vr3, (vr18)
8:  ldci      vr19, $0
9:  mov       vr1, vr19
10: jmp      .L16
```

Assume that vr0, vr1, vr2, and vr3 are live at the end of the basic block.

Show:

1. The virtual registers that are live at the beginning of the block
2. For each instruction in the block, which virtual registers are live at the point *just after* the instruction

Recall that a virtual register is live if it will be used at a later point, but there is not an intervening def (assignment) of the virtual register.

**Question 2.** [25 points] Assume that the following code is a basic block:

```
ldci    vr4, $3
ldci    vr5, $2
mulr    vr6, vr4, vr5
addi    vr7, vr0, vr6
writei   vr7
```

Assume that vr4, vr5, vr6, or vr7 are temporaries, so they are all dead at the end of the block.

(a) Rewrite this code so that all uses of virtual registers with known constant values are replaced with the appropriate constant. For example, in an instruction `mov vr8, vr9`, if vr9 is known to have the constant value 42, then you would rewrite the instruction as `mov vr8, $42`. Note that if a value is computed from constant operands, the computed value should also be treated as a constant (constant folding.)

(b) Which instructions in the transformed code can be eliminated? Explain briefly.

**Question 3.** [20 points] Assume that a local register allocator has three machine registers available, called A, B, and C. For each the following instructions, we want the local register allocator to assign a machine register for each virtual register. The first instruction is annotated to show that `vr0` has been assigned machine register A.

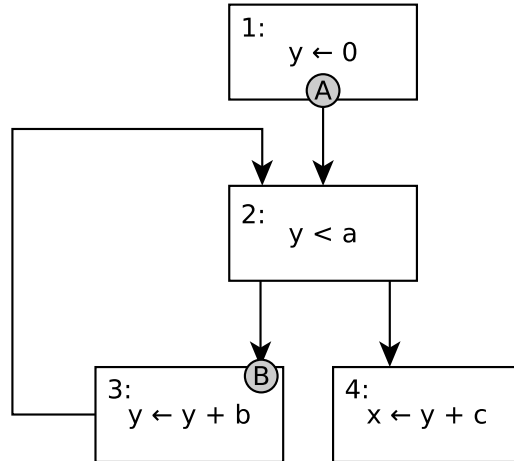
Instruction	Register assignments	Spill and/or restore?
<code>ldci vr0, \$1</code>	<code>vr0</code> →A	
<code>ldci vr1, \$2</code>		
<code>ldci vr2, \$2</code>		
<code>addi vr3, vr0, vr1</code>		
<code>muli vr4, vr3, \$8</code>		
<code>addi vr5, vr4, vr2</code>		

Complete the table above by performing local register allocation. For each instruction in the block, show a possible assignment of machine registers to the referenced virtual registers. Use bottom-up allocation, so that when a spill is necessary, the live virtual register whose next use is furthest in the future is selected as a victim.

Indicate where spills and restores are necessary. You can use the notation `spill VR,MR` and `restore VR,MR`, where VR is the virtual register being spilled or restored, and MR is the machine register assignment being removed by the spill or established by the restore.

Don't forget to reclaim machine registers after the last use of the virtual register to which the machine register is assigned.

**Question 4.** [20 points] Consider the control-flow graph on the right, where block 1 is the entry block and block 4 is the exit block, and the variables are a, b, c, x, and y:



(a) At the point labeled A (end of block 1), which variables are guaranteed to be used on some forward path? (I.e., which variables are guaranteed to have their current values *used* on whichever path to the end of the exit block is executed.)

(b) At the point labeled B (beginning of block 3), which variables are guaranteed to be used on some forward path?

(c) Would a dataflow analysis to find guaranteed uses of variables be a forward analysis or a backward analysis? Explain briefly.

(d) Would a dataflow analysis to find guaranteed uses of variables be a “may” analysis or a “must” analysis? Explain briefly.

**Question 5.** [10 points] Local value numbering (LVN) is useful for detecting redundant computations and replacing them with a use of a previously-computed value.

One potential obstacle to using LVN to eliminate redundant computations is that a storage location (i.e., virtual register) containing a computed value might be overwritten. For example, consider the following code:

```
1:   addi    vr4, vr5, $42
2:   writei  vr4
3:   ldci    vr4, $17
4:   addi    vr6, vr5, $42
5:   writei  vr6
```

At line 4, there is a recomputation of the sum  $vr5 + 42$ , which is a value that was previously stored in `vr4`. However, because `vr4` was modified at line 3, it is not correct to replace the instruction at line 4 with

```
mov      vr6, vr4
```

or to replace the instruction at line 5 with

```
writei    vr4
```

State whether you think this is a significant problem in practice. I.e., is it likely that computed values will become unavailable by being overwritten, or could the compiler guarantee that this will not happen? Hint: think about how code generation for expression evaluation and address computation works. Briefly justify your answer.

[Extra page for answers and/or scratch work.]