ECE 468 & 573

Problem Set 3: Common sub-expression elimination and local register allocation **Solutions**

For the following problems, consider the following piece of three-address code:

```
1.  A = 7;
2.  B = A + 2;
3.  C = A + B;
4.  D = A + B;
5.  A = C + B;
6.  B = C + B;
7.  E = C + D;
8.  F = C + D;
9.  G = A + B;
10. H = E + F;
11. I = H + G;
12. WRITE(I);
```

1. Show the result of performing Common Subexpression Elimination (CSE) on the above code.

   Answer:

   Available expressions shows which expressions are available *after* the code is executed.

Original code	Available expressions	New code
1. A = 7;		A = 7;
2. B = A + 2;	A + 2	B = A + 2
3. C = A + B;	A + 2, A + B	C = A + B
4. D = A + B;	A + 2, A + B	D = C
5. A = C + B;	C + B	A = C + B
6. B = C + B;		B = A
7. E = C + D;	C + D	E = C + D
8. F = C + D;	C + D	F = E
9. G = A + B;	C + D, A + B	G = A + B
10. H = E + F;	C + D, A + B, E + F	H = E + F
11. I = H + G;	C + D, A + B, E + F, H + G	I = H + G
12. WRITE(I);	C + D, A + B, E + F, H + G	WRITE(I)

2. Suppose A and C were aliased. How would that change the results of CSE?

   Answer: If A and C were aliased, then A + B would not be available after statement 3, because writing to C also invalidates any expression that uses A. Thus, statement 4 would still have to be D = A + B. By the same logic, statement 6 would

not be redundant, because statement 5 would kill C + B. On the flip side, Statement 5 could be rewritten to A = D.

3. For each instruction, show which variables are live *immediately after the instruction.*

   Answer: Remember that we start at the bottom, and for each statement, subtract out any variables that are defined and add in any variables that are used.

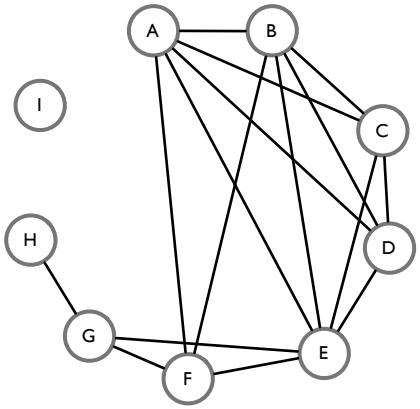
Original code	Variables used	Variables defined	Live variables
1. A = 7;		A	{A}
2. B = A + 2;	A	B	{A, B}
3. C = A + B;	A, B	C	{A, B, C}
4. D = A + B;	A, B	D	{B, C, D}
5. A = C + B;	C, B	A	{A, B, C, D}
6. B = C + B;	C, B	B	{A, B, C, D}
7. E = C + D;	C, D	E	{A, B, C, D, E}
8. F = C + D;	C, D	F	{A, B, E, F}
9. G = A + B;	A, B	G	{G, E, F}
10. H = E + F;	E, F	H	{H, G}
11. I = H + G;	H, G	I	{I}
12. WRITE(I);	I		{ }

4. How many registers would be needed to perform register allocation with no spilling?

   Answer: 5 – there are 5 live variables after instruction 7.

5. Top down register allocation is inefficient for the above code, as there are some variables that could safely be assigned to the same register. What are they?

   Answer: Any two variables that are never live at the same time could share a register. One way to see this is to build the interference graph for this code (as used in graph coloring register allocation). Any two variables that do not share an edge could be allocated to the same register:



6. Perform bottom-up register allocation on the code for a machine with four registers. Show what code would be generated for each 3AC instruction. When choosing registers to allocate, always allocate the lowest-numbered register available. When choosing registers to spill, choose the non-dirty register that will be used farthest in the future. In case all registers are dirty, choose the register that will be used farthest in the future. In case of a tie, choose the lowest-numbered register.

Answer:

After each instruction, I've shown the state of the registers. A * means that the register is dirty.

1. A = 7;
R1 = 7
Register map: R1: A*, R2: , R3: , R4:

2. B = A + 2;
R2 = R1 + 2
Register map: R1: A*, R2: B*, R3: , R4:

3. C = A + B;
R3 = R1 + R2
Register map: R1: A*, R2: B*, R3: C*, R4:

4. D = A + B;
R1 = R1 + R2 //D takes A's place in R1. No need to store since A is dead
Register map: R1: D*, R2: B*, R3: C*, R4:

5. A = C + B;

R4 = R3 + R2;
Register map: R1: D*, R2: B*, R3: C*, R4: A*

6. B = C + B;
R2 = R3 + R2;
Register map: R1: D*, R2: B*, R3: C*, R4: A*

7. E = C + D;
ST R2, B //Tie between A and B to kick out. Need to store because B is dirty
R2 = R3 + R1
Register map: R1: D*, R2: E*, R3: C*, R4: A*

8. F = C + D;
R1 = R3 + R1 //C and D are dead, can free registers and reuse R1
Register map: R1: F*, R2: E*, R3: , R4: A*

9. G = A + B;
LD B, R3 //Need to reload B
R3 = R4 + R3 //A and B dead, can free registers and reuse R3
Register map: R1: F*, R2: E*, R3: G* , R4:

10. H = E + F;
R1 = R2 + R1 //E and F dead, can free registers and reuse R1
Register map: R1: H*, R2: , R3: G* , R4:

11. I = H + G;
R1 = R1 + R3 //H and G dead, can free registers and reuse R1
Register map: R1: I*, R2: , R3: , R4:

12. WRITE(I);
WRITE(R1) //I dead, can free register
Register map: R1: , R2: , R3: , R4:

Repeat steps 3 and 6 for the following code, but with three registers (assume registers can hold either temporaries or variables):

```
1.  T1 = A + B;
2.  T2 = C + D;
```

```
3.  T3 = T1 - T2;
4.  T4 = C + T3;
5.  T5 = D + T3;
6.  T6 = T1 + T5;
7.  T7 = D + T6;
8.  T8  = B + T7;
9.  T9 = A + T4;
10. A  = T1 + T8;
11. T10 = T9 + A;
12. WRITE(T10);
```

Original code	Variables used	Variables defined	Live variables
1. T1 = A + B;	A, B	T1	{T1, A, C, B, D}
2. T2 = C + D;	C, D	T2	{T1, A, C, B, D, T2}
3. T3 = T1 - T2;	T1, T2	T3	{T1, A, C, B, D, T3}
4. T4 = C + T3;	C, T3	T4	{T1, A, T4, B, D, T3}
5. T5 = D + T3;	D, T3	T5	{T1, A, T4, B, D, T5}
6. T6 = T1 + T5;	T1, T5	T6	{T1, A, T4, B, D, T6}
7. T7 = D + T6;	D, T6	T7	{T1, A, T4, B, T7}
8. T8 = B + T7;	B, T7	T8	{T1, T8, A, T4}
9. T9 = A + T4;	A, T4	T9	{T1, T8, T9}
10. A = T1 + T8;	T1, T8	A	{A, T9}
11. T10 = T9 + A;	T9, A	T10	{T10}
12. WRITE(T10);	T10		{ }

Generated code:

1. T1 = A + B;

LD A, R1
LD B, R2
R3 = R1 + R2
Register map: R1: A, R2: B, R3: T1*

2. T2 = C + D;

LD C, R1 //A is farthest away use, so use R1. No store because not dirty
LD D, R2 //B is next farthest away. No store because not dirty
R2 = R1 + R2 //D is farthest away
Register map: R1: C, R2: T2*, R3: T1*

3. T3 = T1 - T2;
R2 = R3 - R2 //T2 is dead, so can reuse R2
Register map: R1: C, R2: T3*, R3: T1*

4. T4 = C + T3;
R1 = R1 + R2 //C is dead, so can reuse R1
Register map: R1: T4*, R2: T3*, R3: T1*

5. T5 = D + T3;
ST R1, T4 //Need register for D. T4 used farthest away. Spill because T4 dirty and live
LD D, R1 //Put D in R1
R2 = R1 + R2 //T3 is dead, so can reuse R2
Register map: R1: D, R2: T5*, R3: T1*

6. T6 = T1 + T5;
R2 = R3 + R2 //T5 is dead, so can reuse R2
Register map: R1: D, R2: T6*, R3: T1*

7. T7 = D + T6;
R1 = R1 + R2; //D and T6 dead, can reuse R1 Register map: R1: T7*, R2: , R3: T1*

8. T8 = B + T7;
LD B, R2 //Put B in R2
R1 = R2 + R1; //B and T7 dead, can reuse R1 Register map: R1: T8*, R2: , R3: T1*

9. T9 = A + T4;
LD A, R2 //Put A in R2
ST R1, T8 //Need register for T4. Tie between T8 and T1. Spill because T8 dirty.
LD T4, R1 //Put T4 in R1
R1 = R2 + R1 //T4 dead, A dead, can reuse R1
Register map: R1: T9*, R2: , R3: T1*

10. A = T1 + T8;
LD T8, R2 //Put T8 in R2
R2 = R3 + R2 //T8, T1 dead, can reuse R2
Register map: R1: T9*, R2: A* , R3:

11. T10 = T9 + A;
R1 = R1 + R2 //T9, A dead, can reuse R1
Register map: R1: T10*, R2: , R3:

12. WRITE(T10);
WRITE(R1) //T10 dead
Register map: R1: , R2: , R3: