CS 335 Semester 2022–2023-II: End-Semester Exam

8–11 AM, April 25th 2023

Suggestions

- There are six pages. Answer questions exactly as asked, and be concise in your explanations.
- Your answers should be legible. Show logical steps and rough work in your computation.
- Explicitly mention ANY ASSUMPTIONS that you made for coming up with your solutions.
- Few questions require you to fill in answers in the question paper itself. Turn in YOUR QUESTION PAPER ALONG WITH YOUR SOLUTION SHEETS.

Questions

1. Answer the following questions related to parsing. **[3×4 marks]**

   (a) Show the left-factored form of the following grammar.

   $$S \to S\dagger \mid S\nabla S \mid S \Updownarrow \mid \lceil T \rceil$$
   $$T \to Ta \mid Tb \mid Tc \mid \epsilon$$

   Answer. Either three marks or zero.

   $$S \to S\alpha \mid \lceil T \rceil$$
   $$\alpha \to \dagger \mid \nabla S \mid \Updownarrow$$
   $$T \to T\beta \mid \epsilon$$
   $$\beta \to a \mid b \mid c$$

   (b) Eliminate left recursion from the following grammar.

   $$S \to Sab \mid S! \mid (T) \mid bTb$$
   $$T \to Ta \mid Tb \mid Tc \mid \epsilon$$

   Answer. Either three marks or zero.

   $$S \to (T)S' \mid bTbS'$$
   $$S' \to abS' \mid !S' \mid \epsilon$$
   $$T \to T'$$
   $$T' \to aT' \mid bT' \mid cT' \mid \epsilon$$

   (c) Consider shift-reduce bottom-up parsers which can support both left and right recursion. Do you think there are any advantages of using left recursion over right recursion in the input grammar for unrestricted input?

Answer. Left recursion can parse a sequence of any number of elements with bounded stack space. For right recursive grammar, the resulting shift-reduce machine will shift the entire input onto the stack, before performing the first reduction. It will accept all strings in the language, but it requires an unbounded stack size.

Giving an example here to explain the problem is fine.

(d) Why do semantic routines need to be placed at the end of productions in LR grammars?

Answer. LR grammars are not predictive; they cannot recognize a production until they see the entire thing. Thus, they cannot apply semantic routines until they have reduced a production, so semantic routines can only go at the end of a production (i.e., at the point where the LR parser will reduce).

2. Consider the following basic block and its constituent instructions. Assume all variables are of type integers, `f` is input to the basic block, and `y` is the only variable live on exit from the basic block.

```
a = f * 2 + 0;
b = a - 0;
c = 1 + 6;
d = c * b;
e = f * f;
x = e + d;
g = b + d;
h = b + d;
i = g * 1;
y = i / h;
```

(i) Apply the following optimizations to the basic block, in order: (i) Algebraic simplification (AS), (ii) Constant folding (CF), (iii) Common sub-expression elimination (CSE), (iv) Strength reduction (SR), and (v) Dead code elimination (DCE). Explain the modifications carried out by each optimization, and show the result of each transformation. The input to optimization i is the output from the optimization $i-1$.

(ii) Can you suggest *additional* optimizations on the final output.

[8 marks]

Answer. The optimization sequences, in order, are as follows.

(i) Algebraic Simplification (AS)

```
a = f * 2;
b = a;
c = 1 + 6;
d = c * b;
e = f * f;
x = e + d;
g = b + d;
h = b + d;
i = g;
y = i / h;
```

(ii) Constant Folding (CF) (Constant propagation is different)

```
a = f * 2;
b = a;
c = 7;
```

```
d = c * b;
e = f * f;
x = e + d;
g = b + d;
h = b + d;
i = g;
y = i / h;
```

Common Sub-Expression Elimination (CSE)

```
a = f * 2;
b = a;
c = 7;
d = c * b;
e = a; f a f;
x = e + d;
g = b + d;
h = g;
i = g;
y = i / h;
```

Strength Reduction (SR)

```
a = f << 1;
b = a;
c = 7;
d = c * b;
e = a; f x f
x = e + d;
g = b + d;
h = g;
i = g;
y = i / h;
```

Dead Code Elimination (DCE)

```
a = f << 1;
b = a;
c = 7;
d = c * b;
g = b + d;
h = g;
i = g;
y = i / h;
```

(ii) The second part is open-ended as students may suggest different optimizations. We can try the following.

```
a = (f << 1);
b = (f << 1); // forward substitution
c = 7;
d = 7 * b; // constant propagation
g = b + d;
h = g;
i = g;
y = i / h;
```

After DCE, the code will look like the following.

```
b = (f << 1); // forward substitution
d = 7 * b; // constant propagation
g = b + d; // can do b + 7*b
h = g;
i = g;
y = i / h;
```

We can further optimize away the variable d.

```
b = (f << 1);
g = 8*b; // Can be simplified to (b<<3)
h = g;
i = g;
y = i / h;
```

We can now remove the variable g.

```
b = (f << 1);
h = (b << 3);
i = h;
y = i / h;
```

3. Consider the following assembly-like IR that makes use of a few temporaries (symbolic registers).

```
t1 = t0 + 15;
t2 = t0 * 3;
if (t1 < t2) {
  t3 = t1 * t2;
  t4 = t1 * 2;
} else {
  t3 = t1 + t2;
  t4 = t1 + t3;
}
t5 = t4 - t2;
t6 = t5 + t3;
t2 = t5 + 1;
t7 = t2 + t3;
if (t6 < t7) {
  t8 = t6;
} else {
  t8 = t7;
}
t9 = t8 * 2;
```

(i) List the temporaries that are live at each program point. Note that `t0` is the only input temporary for the given code and `t9` will be the only live value on exit.

(ii) Draw the interference graph between temporaries for the above program.

(iii) Provide a lower bound on the number of registers required by the program.

(iv) Provide a k-coloring of the interference graph, where k is from item (iii).

[5+3+2+2 marks]

Answer.

(i) List of temporaries that are live at each program point. (5)

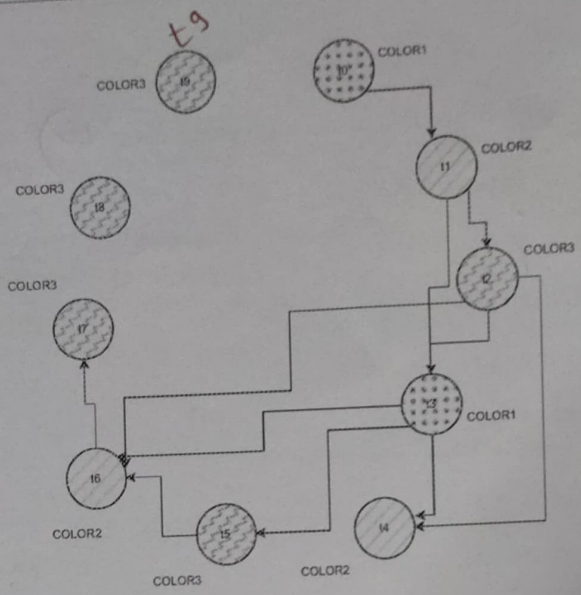
```
1  // Live: t0
2        t1 = t0 + 15;
3  // Live: t0, t1
4        t2 = t0 * 3;
5  // Live: t1, t2
6        if (t1 < t2) {
7  // Live: t1, t2
8          t3 = t1 * t2;
9  // Live: t1, t2, t3
10         t4 = t1 * 2;
11       } else {
12 // Live: t1, t2
13         t3 = t1 + t2;
14 // Live: t1, t2, t3
15         t4 = t1 + t3;
16       }
17 // Live: t2, t3, t4
18       t5 = t4 - t2;
19 // Live: t3, t5
20       t6 = t5 + t3;
21 // Live: t3, t5, t6
22       t2 = t5 + 1;
23 // Live: t2, t3, t6
24       t7 = t2 + t3;
25 // Live: t6, t7
26       if (t6 < t7) {
27 // Live: t6
28         t8 = t6;
29       } else {
30 // Live: t7
31         t8 = t7;
32       }
33 // Live: t8
34       t9 = t8 * 2;
35 // Live: t9
```

(ii) At least three registers will be required as t1, t2, and t3 are live simultaneously. Give one mark if students mention four or more registers are required at minimum.

(ii) graph - 3

(iii) 3 variables - 2

(iv) IG - color - 2



4. Construct an SDT scheme that translates Roman numerals into integers. The grammar and a reference table are given below. Do not worry about numbers that the given grammar cannot generate. **[8 marks]**

$$
\begin{aligned}
\mathit{Rnum} &\rightarrow \mathit{Thousand}\ \mathit{Hundred}\ \mathit{Ten}\ \mathit{Digit} \\
\mathit{Thousand} &\rightarrow \mathrm{M} \mid \mathrm{MM} \mid \epsilon \\
\mathit{Hundred} &\rightarrow \mathit{SmallHundred} \mid \mathrm{CD} \mid \mathrm{D}\ \mathit{Smallhundred} \mid \mathrm{CM} \\
\mathit{SmallHundred} &\rightarrow \mathrm{C} \mid \mathrm{CC} \mid \epsilon \\
\mathit{Ten} &\rightarrow \mathit{SmallTen} \mid \mathrm{XL} \mid \mathrm{L}\ \mathit{SmallTen} \mid \mathrm{XC} \\
\mathit{SmallTen} &\rightarrow \mathrm{X} \mid \mathrm{XX} \mid \epsilon \\
\mathit{Digit} &\rightarrow \mathit{SmallDigit} \mid \mathrm{IV} \mid \mathrm{V}\ \mathit{SmallDigit} \mid \mathrm{IX} \\
\mathit{SmallDigit} &\rightarrow \mathrm{I} \mid \mathrm{II} \mid \mathrm{III} \mid \epsilon
\end{aligned}
$$

Roman numeral	Decimal value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

Answer

Production	Semantic Actions
$Rnum \rightarrow Thousand\ Hundred\ Ten\ Digit$	$\{\ Rnum.val = Thousand.val * 1000 + Hundred.val * 100 + Ten.val * 10 + Digit.val$; print$(Rnum.val)\ \}$
$Thousand \rightarrow$ M	$\{\ Thousand.val = 1\}$
$Thousand \rightarrow$ MM	$\{\ Thousand.val = 2\}$
$Thousand \rightarrow \epsilon$	$\{\ Thousand.val = 0\}$
$Hundred \rightarrow SmallHundred$	$\{\ Hundred.val = SmallHundred.val\ \}$
$Hundred \rightarrow$ CD	$\{\ Hundred.val = 4\ \}$
$Hundred \rightarrow$ D $SmallHundred$	$\{\ Hundred.val = 5 + SmallHundred.val\ \}$
$Hundred \rightarrow$ CM	$\{\ Hundred.val = 9\}$
$SmallHundred \rightarrow$ C	$\{\ SmallHundred.val = 1\}$
$SmallHundred \rightarrow$ CC	$\{\ SmallHundred.val = 2\}$
$smallHundred \rightarrow \epsilon$	$\{\ SmallHundred.val = 0\}$
$Ten \rightarrow smallTen$	$\{\ Ten.val = SmallTen.val\ \}$
$Ten \rightarrow$ XL	$\{\ Ten.val = 4\}$
$Ten \rightarrow$ L $SmallTen$	$\{\ Ten.val = 5 + SmallTen.val\}$
$Ten \rightarrow$ XC	$\{\ Ten.val = 9\}$
$SmallTen \rightarrow$ X	$\{\ SmallTen.val = 1\}$
$SmallTen \rightarrow$ XX	$\{\ SmallTen.val = 2\}$
$SmallTen \rightarrow \epsilon$	$\{\ SmallTen.val = 0\}$
$Digit \rightarrow SmallDigit$	$\{\ Digit.val = SmallDigit.val\ \}$
$Digit \rightarrow$ IV	$\{\ Digit.val = 4\}$
$Digit \rightarrow$ V $SmallDigit$	$\{\ Digit.val = 5 + SmallDigit.val\ \}$
$Digit \rightarrow$ IX	$\{\ Digit.val = 9\}$
$SmallDigit \rightarrow$ I	$\{\ SmallDigit.val = 1\}$
$SmallDigit \rightarrow$ II	$\{\ SmallDigit.val = 2\}$
$SmallDigit \rightarrow$ III	$\{\ SmallDigit.val = 3\}$
$SmallDigit \rightarrow \epsilon$	$\{\ SmallDigit.val = 0\}$

One idea is to work out a few translations to ensure correctness. For example, MMXIV = 2014.[1]

[1] You can check `https://www.calculatorsoup.com/calculators/conversions/roman-numeral-converter.php` for automated calculations.

5. Consider the following grammar with 2 missing productions.

$$S \to aS \mid \ldots(1)$$
$$A \to \ldots(2) \mid \epsilon$$
$$X \to cS \mid \epsilon$$
$$Y \to dS \mid \epsilon$$
$$Z \to eS$$

We know the FIRST and FOLLOW sets for the grammar. Reconstruct the grammar by filling in the TWO missing productions. **[6 marks]**

Symbol	FIRST	FOLLOW
S	FIRST(S) = {a,b,c,d,e}	{\$} ∪ FOLLOW(X) ∪ FOLLOW(Y) ∪ FOLLOW(Z)
A	FIRST(A) = {c,d,e,ϵ}	{b}
X	FIRST(X) = {c,ϵ}	FIRST(Y)/ϵ ∪ FIRST(Z)
Y	FIRST(Y) = {d,ϵ}	FIRST(Z)
Z	FIRST(Z) = {e}	FOLLOW(A)
a	{a}	FIRST(S)
b	{b}	FOLLOW(S)
c	{c}	FIRST(S)
d	{d}	FIRST(S)
e	{e}	FIRST(S)

Answer.

(1) Ab - FIRST(S) has a,b,c,d,e, so (1) should be able to generate b,c,d,e in the first position. Since we have no rule that produces b, it needs to have b, but the b cannot be the first position, since that would not allow another character to be in the first position. FOLLOW(S) ≤ FOLLOW(b), so b would be the last character in this production. To generate c,d,e in the first position we have to use A, since it also has to be able to go to ϵ. FOLLOW(A) = b, so we know there are no other (non-)terminals in between the A and b. (Something like Abbbb would also work, though we did not directly specify b ∈ FOLLOW(b)).

(2) XYZ - Since FIRST(A) has c,d,e, (2) cannot start with a terminal, so we have to use X, Y, and Z to generate c,d,e. Since Z does not go to ϵ and FOLLOW(A) ≤ FOLLOW(Z), we can conclude that Z has to go last. Since FIRST(Y) ≤ FOLLOW(X) and FIRST(Z) ≤ FOLLOW(Y), we can conclude that the order has to be XYZ.

(1) +(2) ✓

0 + (2)

6. Consider the following grammar.

$$S \to aAd \quad (1)$$
$$S \to bBd \quad (2)$$
$$S \to aBe \quad (3)$$
$$S \to bAe \quad (4)$$
$$A \to c \quad (5)$$
$$B \to c \quad (6)$$

[8+5+5+2 marks]

(i) Show the construction of the LR(1) canonical collection of items in your answer sheet. You do not need to draw the LR(1) automaton.

Answer.

Symbol	FIRST	FOLLOW
S	FIRST(S) = {a, b}	FOLLOW(S)= {$}
A	FIRST(A) = {c}	FOLLOW(A) = {d,e}
B	FIRST(B) = {c}	FOLLOW(B)={d,e}

Augmented grammar.

$$S' \to S$$
$$S \to aAd \quad (1)$$
$$S \to bBd \quad (2)$$
$$S \to aBe \quad (3)$$
$$S \to bAe \quad (4)$$
$$A \to c \quad (5)$$
$$B \to c \quad (6)$$

LR(1) canonical collection.

$$I_0 = \text{closure}(S' \to \bullet S)$$
$$= \{S' \to \bullet S, \$$
$$S \to \bullet aAd, \$$
$$S \to \bullet bBd, \$$
$$S \to \bullet aBe, \$$
$$S \to \bullet bAe, \$\}$$

$$I_1 = \text{goto}(I_0, S)$$
$$= \{S' \to S\bullet, \$\}$$

$$I_2 = \text{goto}(I_0, a)$$
$$= \{S \to a \bullet Ad, \$$
$$S \to a \bullet Be, \$$
$$A \to \bullet c, d$$
$$B \to \bullet c, e\}$$

$$I_3 = \text{goto}(I_0, b)$$
$$= \{S \to b \bullet Bd, \$$
$$S \to b \bullet Ae, \$$
$$B \to \bullet c, d$$
$$A \to \bullet c, e\}$$

$$I_4 = \text{goto}(I_2, A)$$
$$= \{S \to aA \bullet d, \$\}$$

$$I_5 = \text{goto}(I_2, B)$$
$$= \{S \to aB \bullet e, \$\}$$

$$I_6 = \text{goto}(I_2, c)$$
$$= \{A \to c\bullet, d$$
$$B \to c\bullet, e\}$$

$$I_7 = \text{goto}(I_3, B)$$
$$= \{S \to bB \bullet d, \$\}$$

$$I_8 = \text{goto}(I_3, A)$$
$$= \{S \to bA \bullet e, \$\}$$

$$I_9 = \text{goto}(I_3, c)$$
$$= \{A \to c\bullet, d$$
$$AB' \to c\bullet, e\}$$

$$I_{10} = \text{goto}(I_4, d)$$
$$= \{S \to aAd\bullet, \$\}$$

$$I_{11} = \text{goto}(I_5, e)$$
$$= \{S \to aBe\bullet, \$\}$$

$$I_{12} = \text{goto}(I_7, d)$$
$$= \{S \to bBd\bullet, \$\}$$

$$I_{13} = \text{goto}(I_8, e)$$
$$= \{S \to bAe\bullet, \$\}$$

There are no conflicts in any state, so the grammar is LR(1).

(ii) Fill in the LR(1) parsing table given below. Add more rows to the table if your collection requires more states. Similarly, ignore additional rows if your collection has fewer states. Use the production RULE NUMBERS AS GIVEN ABOVE for parsing table entries.

Answer.

	Action						Goto		
	a	b	c	d	e	$	S	A	B
0	s2	s3					1		
1						acc			
2			s6					4	5
3			s9					8	7
4				s10					
5					s11				
6				r5	r6				
7				s12					
8					s13				
9				~~r5~~ γ6	~~r6~~ γ5				
10					r1	γ1			
11					r3	γ3			
12					r2	γ2			
13					r4	γ4			

(iii) Fill in the LALR(1) parsing table given below. Add more rows to the table if your collection requires more states. Similarly, ignore additional rows if your collection has fewer states.

Answer.

States 6 and 9 in the LR(1) collection have a common core, so they can be merged. **LR(1) canonical collection.**

$$
\begin{aligned}
I_0 &= \texttt{closure}(S' \to \bullet S) \\
&= \{S' \to \bullet S, \$ \\
&\quad S \to \bullet aAd, \$ \\
&\quad S \to \bullet bBd, \$ \\
&\quad S \to \bullet aBe, \$ \\
&\quad S \to \bullet bAe, \$\}
\end{aligned}
$$

$$
\begin{aligned}
I_1 &= \texttt{goto}(I_0, S) \\
&= \{S' \to S\bullet, \$\}
\end{aligned}
$$

$$
\begin{aligned}
I_2 &= \texttt{goto}(I_0, a) \\
&= \{S \to a \bullet Ad, \$ \\
&\quad S \to a \bullet Be, \$ \\
&\quad A \to \bullet c, d \\
&\quad B \to \bullet c, e\}
\end{aligned}
$$

$$
\begin{aligned}
I_3 &= \texttt{goto}(I_0, b) \\
&= \{S \to b \bullet Bd, \$ \\
&\quad S \to b \bullet Ae, \$ \\
&\quad B \to \bullet c, d \\
&\quad A \to \bullet c, e\}
\end{aligned}
$$

$$
\begin{aligned}
I_4 &= \texttt{goto}(I_2, A) \\
&= \{S \to aA \bullet d, \$\}
\end{aligned}
$$

$$
\begin{aligned}
I_5 &= \texttt{goto}(I_2, B) \\
&= \{S \to aB \bullet e, \$\}
\end{aligned}
$$

$$
\begin{aligned}
I_{6/9} &= \texttt{goto}(I_2, c) = \texttt{goto}(I_3, c) \\
&= \{A \to c\bullet, d/e \\
&\quad B \to c\bullet, e/d\}
\end{aligned}
$$

$$
\begin{aligned}
I_7 &= \texttt{goto}(I_3, B) \\
&= \{S \to bB \bullet d, \$\}
\end{aligned}
$$

$$
\begin{aligned}
I_8 &= \texttt{goto}(I_3, A) \\
&= \{S \to bA \bullet e, \$\}
\end{aligned}
$$

$$
\begin{aligned}
I_{10} &= \texttt{goto}(I_4, d) \\
&= \{S \to aAd\bullet, \$\}
\end{aligned}
$$

$$
\begin{aligned}
I_{11} &= \texttt{goto}(I_5, e) \\
&= \{S \to aBe\bullet, \$\}
\end{aligned}
$$

$$
\begin{aligned}
I_{12} &= \texttt{goto}(I_7, d) \\
&= \{S \to bBd\bullet, \$\}
\end{aligned}
$$

$$
\begin{aligned}
I_{13} &= \texttt{goto}(I_8, e) \\
&= \{S \to bAe\bullet, \$\}
\end{aligned}
$$

LALR(1) Parsing Table.

	Action						Goto		
	a	b	c	d	e	$	S	A	B
0	s2	s3					1		
1						acc			
2			s64					4	5
3			s69					8	7
4				s10					
5					s11				
6/9				r5/r6	r5/r6				
7				s12					
8					s13				
9						r1			
10						r3			
11						r2			
12						r4			

(iv) Briefly explain whether the above grammar is (i) LR(1) and (ii) LALR(1)?

Answer.

There is a reduce/reduce conflict in state I_{69}, so the grammar is not LALR(1).

7. Show an equivalent SSA-form for the following sequence of instructions. Do not forget to include ϕ-functions if needed. [6 marks]

```
        b = 0;
        d = 0;
        a = 1;
        i = ...;
Loop:   if (i > 0) {
          if (a < 0) {
            b = i;
          } else {
            b = 0;
          }
          i = i - 1;
          if (i < 0) {
            i = 0;
            goto Break;
          }
          if (b == 0) {
            goto Break;
          } else {
            a = a + d;
          }
          goto Loop;
        }
Break:  x = a;
        y = b;
```

Answer.

```
        b1 = 0;
        d1 = 0;
        a1 = 1;
        i1 = ...;

Loop:   i3 = φ(i1, i2)
        a3 = φ(a1, a2)
        b3 = φ(b1, b6)
        if (i3 > 0) {
          if (a3 < 0) {
            b4 = i;
          } else {
            b5 = 0;
          }
          b6 = φ(b4,b5)
          i2 = i3 - 1;
          if (i2 < 0) {
            i4 = 0;
            goto Break;
          }
          if (b6 == 0) {
            goto Break;
          } else {
            a2 = a3 + d1;
          }
          goto Loop;
        }
Break:  b7 = φ(b3,b6)
```

```
x1 = a3;
y1 = b7;
```

8. Consider the intermediate code given below. [2+2+2+2 marks]

```
1   i = 1
2   j = 1
3   t1 = 5 * i
4   t2 = t1 + j
5   t3 = 4 * t2
6   t4 = t3
7   a[t4] = -1
8   j = j + 1
9   if j <= 5 goto (3)
10  i = i + 1
11  if i < 5 goto (2)
```

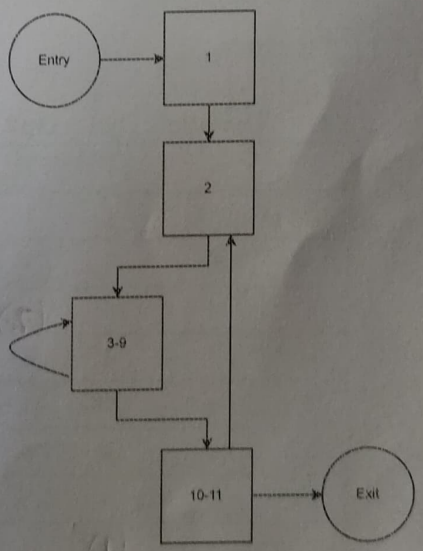
The symbols `a`, `i`, and `j` refer to variables in the program, while names like `t1` refer to temporaries.

(i) Identify the basic blocks using instruction numbers. Draw the control flow graph.

(ii) Write the quadruple and triple form of the code snippet.

Answer.

Leader instructions: (1), (2), (3), and (10). So, BBs are: (1),(2), (3)–(9), (10)–(11).



id 3
b 3
a 3
c 1

5) 6

2) 5+2=7

8) 4+
ii) 2 =6

7) 2

6) 3+2=5

36

	Opcode	Op1	Op2	Result
1	=	1		i
2	=	1		j
3	*	5	i	t1
4	+	t1	j	t2
5	*	4	t2	t3
6	=	t3		t4
7a	[]	a	t4	t99
7b	=	-1		t99
8	+	j	1	j
9	<=	j	5	(3)
10	+	i	1	i
11	<	i	5	(2)

	Opcode	Op1	Op2	
1	=	i	1	i = 1
2	=	j	1	j = 1
3	*	5	(1)	t1 = 5 * i
4	+	(3)	(2)	t2 = t1 + j
5	*	4	(4)	t3 = 4 * t2
6	=	(5)		t4 = t3
7a	[]	a	i	a[t4] = -1
7b	=	(7a)	-1	
8	+	(1)	1	j = j + 1
9a	<=	(2)	5	j <= 5
9b	CCSET	(3)		goto 3
10	+	(1)	1	i = i + 1
11a	<	(i)	5	i < 5
11b	CCSET	(2)		goto 2