

CS39003 Compilers Laboratory
Autumn 2026
Assignment 9
Date of posting: 13-Oct-2026

Intermediate code generation with arrays

This assignment (LA9) is an extension of Lab Assignment LA8. Whatever your program can do on inputs consistent with the grammar of LA8 will also be parsed identically by your program for LA9. There are a few modifications that you need to make. These modifications are explained now.

Storage of 3-address instructions in quadruples (quads)

In LA8, you generated 3-address instructions, printed them, and forgot them in the rest of your code. In LA9, you do not print the 3-address instructions as soon as they are generated. Instead you store them in a table **QT** of quads. After the entire input file is parsed, you print all the quads stored in **QT** in the same sequence as they are generated.

A quad consists of four fields: **Operation**, **Argument1**, **Argument2**, and **Result**. In the last three fields, store the address pointers that you use during parsing.

- Operation** can be a string. Unlike in LA8, you now store detailed information in **Operation**. For an arithmetic operation (like **+**), you now additionally store the type of the operands (must be the same). Recall that Mathorola CPU's supply four basic types: **int**, **rational**, **float**, and **complex**. Use the first letter of the operand type in **Operation**. For example, the arithmetic operations on integers are stored in **Operation** as **iadd**, **isub**, **imul**, **idiv**, and **irem**. For complex numbers, these are **cadd**, **csub**, **cmul**, and **cdiv** (complex numbers have no remainder operation). For a 3-address instruction doing type conversion, **Operation** stores the same format **src2dst** as explained in LA8. Likewise, the constructor-type instructions **mkrtl** and **mkcpx** are stored as **Operation**.
- A reference to a variable in the r-value of an assignment will still be referred to by the name of the variable. However, the storage of an r-value to a variable will use the special **Operation** called **pstr**, where the prefix **p** denotes the type of the r-value (**i**, **r**, **f**, or **c**). Recall that a variable gets the type of the r-value irrespective of whether it is defined or not (array elements may be an exception; see below). Examples: **istr**, **rstr**, **fstr**, **cstr**.
- For initializing a temporary to a constant, use the **Operation** **pset** (**p** is the type prefix). Example: **iset t32 0**.
- We will use arrays (see below) that can be both an l-value or an operand in an r-value. If an array element is used as an operand in an r-value, use **pgetarr** as **Operation** (**p** is the prefix for the type of the array element). For storing an element in an array, use **pstrarr** as **Operation**. For a **pgetarr** instruction, **Argument1** is the array name, and **Argument2** is the offset (in bytes), and the result is a temporary. For a **pstrarr** instruction, **Argument1** is a temporary (address of the value to store), and **Argument2** is the offset (in bytes), and the result is the array name. Examples: **t30 = A[t27]** is stored as **fgetarr A t27 t30**, and **C[t53] = t47** as **cstrarr t47 t53 C**.
- Jump instructions have the following **Operation** codes. An unconditional jump has the code **jmp**. A conditional jump based on a relation **rel** (one of **eq**, **ne**, **gt**, **ge**, **lt**, **le**) has the code **jmprel**. Example: **jmpgt a 10 1234**.
- Quad numbering starts from 1000. The jump labels are the quad numbers (not symbolic labels).

Modifications in the Symbol Table (ST)

Each ST-entry would now contain the name of a variable (a C-style variable name), the type of a variable (one of the basic types **INT**, **RTL**, **FLT**, **CPX** or the array type **ARR**). For an array-type variable, the type of each element of the array should be stored in the ST-entry of that variable. We will use only one-dimensional arrays, so this element type will be a basic type.

Use of one-dimensional arrays

Support for 1-d arrays should be added. **MathC** arrays will follow the conventions elaborated below. Let us call the variables of basic data types as *scalars*.

- **Array types:** Like scalars, arrays are not declared beforehand. The context will make it clear whether we are referring to a scalar or an array element. A variable can dynamically change from a scalar type to an array type, and conversely. However, when a variable refers to an array, its element type will remain fixed (so long as the variable stands for that array). Since variables can assume any types (scalars and even arrays), the offsets of the variables cannot be computed by the compiler. Moreover, the absence of an array size makes it impossible for the compiler to determine its width. Allocation of memory to variables will be the job of the run-time environment. However, every element of an array will be of a fixed type (during its existence as the current array), and array indexing is assumed to be zero-based. So the offset of an index relative to the base of the array can be computed by 3-address instructions.

Here is an example of a variable of the same name with different types at different times.

```

A = 5 + 6;           /* A is now an integer scalar */
A = 5 // 6;          /* A is now a rational scalar */
A[5] = 777 % 8.25;    /* A is now an array of integers */
A[11] = 7.89 @ 9.87;  /* A is still an array of integers */
A = 1.23456789;       /* A is now a floating-point scalar */
A[11] = 7.89 @ 9.87;  /* A is now an array of complex numbers */
A[5] = 777 % 8.25;    /* A is still an array of complex numbers */

```

- **Array references:** Like scalar **ID**'s, array elements can be used as scalar operands in expressions and also as components in rational and complex constructs. An array element is referred to as **A[e]**, where **A** is an **ID**, and **e** is an expression. An array index must be of type **int**, so if **e** evaluates to a value of another basic type, it is typecast to **int** before it is used as an index in **A**.

- If **A** is not defined, or the current type of **A** is not **ARR**, then the *integer* 0 is used for **A[e]**. If **A** is undefined, no new ST entry for **A** is to be created.
- If **A** is defined as an array of element type **t** (a basic type), then **A[e]** will be of type **t**, and has an offset of (the possibly typecast) **e** times the width of **t**, relative to the base of **A**. Use the following widths (in bytes) for Mathorola CPU's: **width(int) = 4**, **width(rtl) = 8**, **width(flt) = 8**, and **width(cpx) = 16**.

The last instruction for accessing **A[e]** will be a **pgetarr Operation** (**p** is the usual type prefix). Notice that the compiler cannot determine whether **A[e]** has been assigned a value earlier (because the compiler does not evaluate **e** during compilation; it only generates the three-address code for **e**). It will be the duty of the run-time environment to manage and supply values of array elements. In run-time, an access to an undefined **A[e]** may evaluate to 0 or an uninitialized value, or may lead to a segmentation fault.

- **Assignments to array elements:** This pertains to the case when an array element **A[e]** is the l-value of an assignment. As above, the code for **e** is generated, and if the type of **e** is not **int**, it is typecast to **int**.
 - If **A** is undefined or is defined but is of a scalar type, the ST entry for **A** will now be an **ARR** of element type the same as the type of the r-value. Then the r-value is stored at the offset (to be computed) of **A**. The storage **Operation** will be **pstrarr** (**p** is the usual type prefix of the r-value).
 - If **A** is currently defined as an array of element type **t**, then the r-value, if not of type **t**, will be first converted to type **t**. The r-value is then stored at the offset (to be computed) using the **Operation pstrarr**.
- **Copying bands of array elements:** This high-level assignment statement copies a band of elements from a source array **S** to a destination array **D**. Since arrays do not have pre-defined sizes, copying an entire array to another is not an option. Instead **MathC** programmers can use the two following assignment statements.

```

D = S[first to last];
D + i = S[first to last];

```

In both the cases, **to** is a keyword, and the beginning index **first** and the ending index **last** are expressions. If necessary, these are typecast to **int**. The first copy-band statement copies the array elements **S[first]**, **S[first + 1]**, **S[first + 2]**, ..., **S[last]** to **D[0]**, **D[1]**, **D[2]**, ..., **D[last - first]**, respectively. For the second copy-band statement, the destination locations are respectively **D[i]**, **D[i + 1]**, **D[i + 2]**, ..., **D[i + last - first]**.

- If **S** is undefined or is defined but is of a scalar type, this copy is not permitted. The user is warned about that (print an error message), and no copying code is generated.
- If **S** is currently defined as an array of element type **stype**, then the symbol-table entry for the destination array **D** is looked at.

- If **D** is undefined or is defined but is of a scalar type, the ST entry for **D** will now be **ARR** of basic element type **dtype** = **stype**.
- If **D** is currently defined as an array of element type **dtype**, then this type is remembered.

Now, an element-by-element copy will be made from **S** to **D**. If the types **stype** and **dtype** are different, then every element read from **S** is typecast to **dtype** before storing in **D**. The element-by-element copy from **S** to **D** is carried out by a pre-test loop. This loop has the following pseudocode for the statement: **D + i = S[first to last]**; Notice that if **first** evaluates to a value larger than **last** (in run time), then no copy is made.

```

    Let stype = S.element.type, and swidth = S.element.width.
    Let dtype = D.element.type, and dwidth = D.element.width.
    Doffset = i * dwidth.          /* Doffset is a temporary */
    Soffset = first * swidth.      /* Soffset is a temporary */
    Slast = last * swidth.        /* Slast is a temporary */
BEGIN: if Soffset > Slast, goto END.
    Selement = S [ Soffset ].     /* Selement is a temporary */
    If (dtype ≠ stype) Selement = stype2dtype Selement.
    D [ Doffset ] = Selement.
    Soffset = Soffset + swidth.
    Doffset = Doffset + dwidth.
    Goto BEGIN.
END:

```

For the statement **D = S[first to last]**, **Doffset** is to be initialized to 0 (use **iset**). The rest of the code will be the same as above. In this code, **BEGIN** and **END** are quad numbers (not symbolic labels). A simple type of backpatching is needed. The quad number **BEGIN** is remembered for the unconditional jump at the end of the loop. The label **END** is not known beforehand, so the jump address for the conditional jump at label **BEGIN** is not available at the time the quad is generated. Later, when **END** is available, this jump address is stored at the quad at index **BEGIN** in the quad table **QT**. This procedure justifies the storage of 3-address instructions (instead of immediately printing them at the time they are generated).

What to write and submit

Modify your code for LA8 as follows.

- Write a new source file for quad utilities (for printing them; see the Sample Output for the printing format).
- Modify the lex file to return additional tokens: the keyword **to**, and the array index or index-range delimiters **[** and **]**.
- Modify the yacc file of LA8: Replace printing 3-address instructions by storing them in **QT** (starting from index 1000).
- Add the following new productions to the yacc file to handle arrays.

```

ASGN    →    ID [ EXPR ] = EXPR ;
ASGN    →    ID = ID [ EXPR to EXPR ] ;
ASGN    →    ID + EXPR = ID [ EXPR to EXPR ] ;
FACTOR  →    ID [ EXPR ]
RCOMP   →    ID [ EXPR ]

```

- Update your C/C++ file for symbol-table utilities to handle the storage of array-element types.
- Update your C/C++ file for address-related utilities to take care of the new productions in the grammar.
- Update your earlier **main()** function to print all stored quads **after** **yyparse()** returns. Use the printing format as in the Sample Output. Only if time permits, add the headers before the code of each assignment statement, and align the quad components as in the sample. You must follow this sequence in the printing of each quad:

```

Operation  Argument1  Argument2  Result

```

- Update the Makefile (as demanded by the add-ons).
- Club together all your source files (do not include any machine-generated file) in a single tar/tgz/zip archive. Submit that archive.

Sample Output

Input	Output																																																																																															
A = 5 + 6;	<table><tr><td></td><td>Operation</td><td>Argument 1</td><td>Argument 2</td><td>Result</td></tr><tr><td>1000:</td><td>iadd</td><td>5</td><td>6</td><td>t1</td></tr><tr><td>1001:</td><td>istr</td><td>t1</td><td></td><td>A</td></tr></table>		Operation	Argument 1	Argument 2	Result	1000:	iadd	5	6	t1	1001:	istr	t1		A																																																																																
	Operation	Argument 1	Argument 2	Result																																																																																												
1000:	iadd	5	6	t1																																																																																												
1001:	istr	t1		A																																																																																												
A = 5 // 6;	<table><tr><td></td><td>Operation</td><td>Argument 1</td><td>Argument 2</td><td>Result</td></tr><tr><td>1002:</td><td>mkrtl</td><td>5</td><td>6</td><td>t2</td></tr><tr><td>1003:</td><td>rstr</td><td>t2</td><td></td><td>A</td></tr></table>		Operation	Argument 1	Argument 2	Result	1002:	mkrtl	5	6	t2	1003:	rstr	t2		A																																																																																
	Operation	Argument 1	Argument 2	Result																																																																																												
1002:	mkrtl	5	6	t2																																																																																												
1003:	rstr	t2		A																																																																																												
A[5] = 777 % 8.25;	<table><tr><td></td><td>Operation</td><td>Argument 1</td><td>Argument 2</td><td>Result</td></tr><tr><td>1004:</td><td>flt2int</td><td>8.250000</td><td></td><td>t3</td></tr><tr><td>1005:</td><td>irem</td><td>777</td><td>t3</td><td>t4</td></tr><tr><td>1006:</td><td>imul</td><td>5</td><td>4</td><td>t5</td></tr><tr><td>1007:</td><td>istrarr</td><td>t4</td><td>t5</td><td>A</td></tr></table>		Operation	Argument 1	Argument 2	Result	1004:	flt2int	8.250000		t3	1005:	irem	777	t3	t4	1006:	imul	5	4	t5	1007:	istrarr	t4	t5	A																																																																						
	Operation	Argument 1	Argument 2	Result																																																																																												
1004:	flt2int	8.250000		t3																																																																																												
1005:	irem	777	t3	t4																																																																																												
1006:	imul	5	4	t5																																																																																												
1007:	istrarr	t4	t5	A																																																																																												
A[11] = 7.89 @ 9.87;	<table><tr><td></td><td>Operation</td><td>Argument 1</td><td>Argument 2</td><td>Result</td></tr><tr><td>1008:</td><td>mkcpx</td><td>7.890000</td><td>9.870000</td><td>t6</td></tr><tr><td>1009:</td><td>cpx2int</td><td>t6</td><td></td><td>t7</td></tr><tr><td>1010:</td><td>imul</td><td>11</td><td>4</td><td>t8</td></tr><tr><td>1011:</td><td>istrarr</td><td>t7</td><td>t8</td><td>A</td></tr></table>		Operation	Argument 1	Argument 2	Result	1008:	mkcpx	7.890000	9.870000	t6	1009:	cpx2int	t6		t7	1010:	imul	11	4	t8	1011:	istrarr	t7	t8	A																																																																						
	Operation	Argument 1	Argument 2	Result																																																																																												
1008:	mkcpx	7.890000	9.870000	t6																																																																																												
1009:	cpx2int	t6		t7																																																																																												
1010:	imul	11	4	t8																																																																																												
1011:	istrarr	t7	t8	A																																																																																												
A = 1.23456789;	<table><tr><td></td><td>Operation</td><td>Argument 1</td><td>Argument 2</td><td>Result</td></tr><tr><td>1012:</td><td>fstr</td><td>1.234568</td><td></td><td>A</td></tr></table>		Operation	Argument 1	Argument 2	Result	1012:	fstr	1.234568		A																																																																																					
	Operation	Argument 1	Argument 2	Result																																																																																												
1012:	fstr	1.234568		A																																																																																												
A[11] = 7.89 @ 9.87;	<table><tr><td></td><td>Operation</td><td>Argument 1</td><td>Argument 2</td><td>Result</td></tr><tr><td>1013:</td><td>mkcpx</td><td>7.890000</td><td>9.870000</td><td>t9</td></tr><tr><td>1014:</td><td>imul</td><td>11</td><td>16</td><td>t10</td></tr><tr><td>1015:</td><td>cstrarr</td><td>t9</td><td>t10</td><td>A</td></tr></table>		Operation	Argument 1	Argument 2	Result	1013:	mkcpx	7.890000	9.870000	t9	1014:	imul	11	16	t10	1015:	cstrarr	t9	t10	A																																																																											
	Operation	Argument 1	Argument 2	Result																																																																																												
1013:	mkcpx	7.890000	9.870000	t9																																																																																												
1014:	imul	11	16	t10																																																																																												
1015:	cstrarr	t9	t10	A																																																																																												
A[5] = 777 % 8.25;	<table><tr><td></td><td>Operation</td><td>Argument 1</td><td>Argument 2</td><td>Result</td></tr><tr><td>1016:</td><td>flt2int</td><td>8.250000</td><td></td><td>t11</td></tr><tr><td>1017:</td><td>irem</td><td>777</td><td>t11</td><td>t12</td></tr><tr><td>1018:</td><td>int2cpx</td><td>t12</td><td></td><td>t13</td></tr><tr><td>1019:</td><td>imul</td><td>5</td><td>16</td><td>t14</td></tr><tr><td>1020:</td><td>cstrarr</td><td>t13</td><td>t14</td><td>A</td></tr></table>		Operation	Argument 1	Argument 2	Result	1016:	flt2int	8.250000		t11	1017:	irem	777	t11	t12	1018:	int2cpx	t12		t13	1019:	imul	5	16	t14	1020:	cstrarr	t13	t14	A																																																																	
	Operation	Argument 1	Argument 2	Result																																																																																												
1016:	flt2int	8.250000		t11																																																																																												
1017:	irem	777	t11	t12																																																																																												
1018:	int2cpx	t12		t13																																																																																												
1019:	imul	5	16	t14																																																																																												
1020:	cstrarr	t13	t14	A																																																																																												
B[0] = 1.234 // A[5];	<table><tr><td></td><td>Operation</td><td>Argument 1</td><td>Argument 2</td><td>Result</td></tr><tr><td>1021:</td><td>imul</td><td>5</td><td>16</td><td>t15</td></tr><tr><td>1022:</td><td>cgetarr</td><td>A</td><td>t15</td><td>t16</td></tr><tr><td>1023:</td><td>flt2int</td><td>1.234000</td><td></td><td>t17</td></tr><tr><td>1024:</td><td>cpx2int</td><td>t16</td><td></td><td>t18</td></tr><tr><td>1025:</td><td>mkrtl</td><td>t17</td><td>t18</td><td>t19</td></tr><tr><td>1026:</td><td>imul</td><td>0</td><td>8</td><td>t20</td></tr><tr><td>1027:</td><td>rstrarr</td><td>t19</td><td>t20</td><td>B</td></tr></table>		Operation	Argument 1	Argument 2	Result	1021:	imul	5	16	t15	1022:	cgetarr	A	t15	t16	1023:	flt2int	1.234000		t17	1024:	cpx2int	t16		t18	1025:	mkrtl	t17	t18	t19	1026:	imul	0	8	t20	1027:	rstrarr	t19	t20	B																																																							
	Operation	Argument 1	Argument 2	Result																																																																																												
1021:	imul	5	16	t15																																																																																												
1022:	cgetarr	A	t15	t16																																																																																												
1023:	flt2int	1.234000		t17																																																																																												
1024:	cpx2int	t16		t18																																																																																												
1025:	mkrtl	t17	t18	t19																																																																																												
1026:	imul	0	8	t20																																																																																												
1027:	rstrarr	t19	t20	B																																																																																												
A[6] = ((1 + 2 * 3) * 4 + (5 * 6 % 7) - 8) / A[5] ;	<table><tr><td></td><td>Operation</td><td>Argument 1</td><td>Argument 2</td><td>Result</td></tr><tr><td>1028:</td><td>imul</td><td>2</td><td>3</td><td>t21</td></tr><tr><td>1029:</td><td>iadd</td><td>1</td><td>t21</td><td>t22</td></tr><tr><td>1030:</td><td>imul</td><td>t22</td><td>4</td><td>t23</td></tr><tr><td>1031:</td><td>imul</td><td>5</td><td>6</td><td>t24</td></tr><tr><td>1032:</td><td>irem</td><td>t24</td><td>7</td><td>t25</td></tr><tr><td>1033:</td><td>iadd</td><td>t23</td><td>t25</td><td>t26</td></tr><tr><td>1034:</td><td>isub</td><td>t26</td><td>8</td><td>t27</td></tr><tr><td>1035:</td><td>imul</td><td>5</td><td>16</td><td>t28</td></tr><tr><td>1036:</td><td>cgetarr</td><td>A</td><td>t28</td><td>t29</td></tr><tr><td>1037:</td><td>int2cpx</td><td>t27</td><td></td><td>t30</td></tr><tr><td>1038:</td><td>cdiv</td><td>t30</td><td>t29</td><td>t31</td></tr><tr><td>1039:</td><td>imul</td><td>6</td><td>16</td><td>t32</td></tr><tr><td>1040:</td><td>cstrarr</td><td>t31</td><td>t32</td><td>A</td></tr></table>		Operation	Argument 1	Argument 2	Result	1028:	imul	2	3	t21	1029:	iadd	1	t21	t22	1030:	imul	t22	4	t23	1031:	imul	5	6	t24	1032:	irem	t24	7	t25	1033:	iadd	t23	t25	t26	1034:	isub	t26	8	t27	1035:	imul	5	16	t28	1036:	cgetarr	A	t28	t29	1037:	int2cpx	t27		t30	1038:	cdiv	t30	t29	t31	1039:	imul	6	16	t32	1040:	cstrarr	t31	t32	A																									
	Operation	Argument 1	Argument 2	Result																																																																																												
1028:	imul	2	3	t21																																																																																												
1029:	iadd	1	t21	t22																																																																																												
1030:	imul	t22	4	t23																																																																																												
1031:	imul	5	6	t24																																																																																												
1032:	irem	t24	7	t25																																																																																												
1033:	iadd	t23	t25	t26																																																																																												
1034:	isub	t26	8	t27																																																																																												
1035:	imul	5	16	t28																																																																																												
1036:	cgetarr	A	t28	t29																																																																																												
1037:	int2cpx	t27		t30																																																																																												
1038:	cdiv	t30	t29	t31																																																																																												
1039:	imul	6	16	t32																																																																																												
1040:	cstrarr	t31	t32	A																																																																																												
C[A[6]] = ((1 + 2 * 3) * 4 + (A[5.] * 6 % 7) - 8) / 9 ;	<table><tr><td></td><td>Operation</td><td>Argument 1</td><td>Argument 2</td><td>Result</td></tr><tr><td>1041:</td><td>imul</td><td>6</td><td>16</td><td>t33</td></tr><tr><td>1042:</td><td>cgetarr</td><td>A</td><td>t33</td><td>t34</td></tr><tr><td>1043:</td><td>imul</td><td>2</td><td>3</td><td>t35</td></tr><tr><td>1044:</td><td>iadd</td><td>1</td><td>t35</td><td>t36</td></tr><tr><td>1045:</td><td>imul</td><td>t36</td><td>4</td><td>t37</td></tr><tr><td>1046:</td><td>flt2int</td><td>5.000000</td><td></td><td>t38</td></tr><tr><td>1047:</td><td>imul</td><td>t38</td><td>16</td><td>t39</td></tr><tr><td>1048:</td><td>cgetarr</td><td>A</td><td>t39</td><td>t40</td></tr><tr><td>1049:</td><td>int2cpx</td><td>6</td><td></td><td>t41</td></tr><tr><td>1050:</td><td>cmul</td><td>t40</td><td>t41</td><td>t42</td></tr><tr><td>1051:</td><td>cpx2int</td><td>t42</td><td></td><td>t43</td></tr><tr><td>1052:</td><td>irem</td><td>t43</td><td>7</td><td>t44</td></tr><tr><td>1053:</td><td>iadd</td><td>t37</td><td>t44</td><td>t45</td></tr><tr><td>1054:</td><td>isub</td><td>t45</td><td>8</td><td>t46</td></tr><tr><td>1055:</td><td>idiv</td><td>t46</td><td>9</td><td>t47</td></tr><tr><td>1056:</td><td>cpx2int</td><td>t34</td><td></td><td>t48</td></tr><tr><td>1057:</td><td>imul</td><td>t48</td><td>4</td><td>t49</td></tr><tr><td>1058:</td><td>istrarr</td><td>t47</td><td>t49</td><td>C</td></tr></table>		Operation	Argument 1	Argument 2	Result	1041:	imul	6	16	t33	1042:	cgetarr	A	t33	t34	1043:	imul	2	3	t35	1044:	iadd	1	t35	t36	1045:	imul	t36	4	t37	1046:	flt2int	5.000000		t38	1047:	imul	t38	16	t39	1048:	cgetarr	A	t39	t40	1049:	int2cpx	6		t41	1050:	cmul	t40	t41	t42	1051:	cpx2int	t42		t43	1052:	irem	t43	7	t44	1053:	iadd	t37	t44	t45	1054:	isub	t45	8	t46	1055:	idiv	t46	9	t47	1056:	cpx2int	t34		t48	1057:	imul	t48	4	t49	1058:	istrarr	t47	t49	C
	Operation	Argument 1	Argument 2	Result																																																																																												
1041:	imul	6	16	t33																																																																																												
1042:	cgetarr	A	t33	t34																																																																																												
1043:	imul	2	3	t35																																																																																												
1044:	iadd	1	t35	t36																																																																																												
1045:	imul	t36	4	t37																																																																																												
1046:	flt2int	5.000000		t38																																																																																												
1047:	imul	t38	16	t39																																																																																												
1048:	cgetarr	A	t39	t40																																																																																												
1049:	int2cpx	6		t41																																																																																												
1050:	cmul	t40	t41	t42																																																																																												
1051:	cpx2int	t42		t43																																																																																												
1052:	irem	t43	7	t44																																																																																												
1053:	iadd	t37	t44	t45																																																																																												
1054:	isub	t45	8	t46																																																																																												
1055:	idiv	t46	9	t47																																																																																												
1056:	cpx2int	t34		t48																																																																																												
1057:	imul	t48	4	t49																																																																																												
1058:	istrarr	t47	t49	C																																																																																												
A[7] = ((A[(1 + 2) * 3]) * 4 + (5 * 6 % 7) - 8.) / 9 ;	<table><tr><td></td><td>Operation</td><td>Argument 1</td><td>Argument 2</td><td>Result</td></tr><tr><td>1059:</td><td>iadd</td><td>1</td><td>2</td><td>t50</td></tr><tr><td>1060:</td><td>imul</td><td>t50</td><td>3</td><td>t51</td></tr><tr><td>1061:</td><td>imul</td><td>t51</td><td>16</td><td>t52</td></tr></table>		Operation	Argument 1	Argument 2	Result	1059:	iadd	1	2	t50	1060:	imul	t50	3	t51	1061:	imul	t51	16	t52																																																																											
	Operation	Argument 1	Argument 2	Result																																																																																												
1059:	iadd	1	2	t50																																																																																												
1060:	imul	t50	3	t51																																																																																												
1061:	imul	t51	16	t52																																																																																												

C = A[5 to 7];

C + 3 = C[0 to 2];

C + 6 = B[0 to 2 // 3];

D + C[1] % C[0] =
 B[10 - A[5] + A[A[6]]
 to (A[5] + C[A[6]+D[8]]) * 20 / 6];

1062:	cgetarr	A	t52	t53
1063:	int2cpx	4		t54
1064:	cmul	t53	t54	t55
1065:	imul	5	6	t56
1066:	irem	t56	7	t57
1067:	int2cpx	t57		t58
1068:	cadd	t55	t58	t59
1069:	flt2cpx	8.000000		t60
1070:	csub	t59	t60	t61
1071:	int2cpx	9		t62
1072:	cdiv	t61	t62	t63
1073:	imul	7	16	t64
1074:	cstrarr	t63	t64	A

	Operation	Argument 1	Argument 2	Result
1075:	iset	0		t65
1076:	imul	5	16	t66
1077:	imul	7	16	t67
1078:	jmpgt	t66	t67	1085
1079:	cgetarr	A	t66	t68
1080:	cpx2int	t68		t69
1081:	istrarr	t69	t65	C
1082:	iadd	t65	4	t65
1083:	iadd	t66	16	t66
1084:	jmp			1078

	Operation	Argument 1	Argument 2	Result
1085:	imul	3	4	t70
1086:	imul	0	4	t71
1087:	imul	2	4	t72
1088:	jmpgt	t71	t72	1094
1089:	igetarr	C	t71	t73
1090:	istrarr	t73	t70	C
1091:	iadd	t70	4	t70
1092:	iadd	t71	4	t71
1093:	jmp			1088

	Operation	Argument 1	Argument 2	Result
1094:	mkrtl	2	3	t74
1095:	imul	6	4	t75
1096:	imul	0	8	t76
1097:	rtl2int	t74		t77
1098:	imul	t77	8	t78
1099:	jmpgt	t76	t78	1106
1100:	rgetarr	B	t76	t79
1101:	rtl2int	t79		t80
1102:	istrarr	t80	t75	C
1103:	iadd	t75	4	t75
1104:	iadd	t76	8	t76
1105:	jmp			1099

	Operation	Argument 1	Argument 2	Result
1106:	imul	1	4	t81
1107:	igetarr	C	t81	t82
1108:	imul	0	4	t83
1109:	igetarr	C	t83	t84
1110:	irem	t82	t84	t85
1111:	imul	5	16	t86
1112:	cgetarr	A	t86	t87
1113:	int2cpx	10		t88
1114:	csub	t88	t87	t89
1115:	imul	6	16	t90
1116:	cgetarr	A	t90	t91
1117:	cpx2int	t91		t92
1118:	imul	t92	16	t93
1119:	cgetarr	A	t93	t94
1120:	cadd	t89	t94	t95
1121:	imul	5	16	t96
1122:	cgetarr	A	t96	t97
1123:	imul	6	16	t98
1124:	cgetarr	A	t98	t99
1125:	int2cpx	0		t100
1126:	cadd	t99	t100	t101
1127:	cpx2int	t101		t102
1128:	imul	t102	4	t103
1129:	igetarr	C	t103	t104
1130:	int2cpx	t104		t105
1131:	cadd	t97	t105	t106
1132:	int2cpx	20		t107
1133:	cmul	t106	t107	t108
1134:	int2cpx	6		t109
1135:	cdiv	t108	t109	t110
1136:	imul	t85	8	t111
1137:	cpx2int	t95		t112
1138:	imul	t112	8	t113
1139:	cpx2int	t110		t114
1140:	imul	t114	8	t115
1141:	jmpgt	t113	t115	1147
1142:	rgetarr	B	t113	t116
1143:	rstrarr	t116	t111	D
1144:	iadd	t111	8	t111
1145:	iadd	t113	8	t113
1146:	jmp			1141

	Operation	Argument 1	Argument 2	Result
--	-----------	------------	------------	--------

1147: