Introduction to Compilers
Fall 2020 Final Exam

(January 15/6pm to 17/11pm, 2021, Take Home Exam)

B07902048 資工三 李宥霆

Note:

*Take home exam must be **Private** and **Independent** work, no consultations or discussions with others*

1) The total score is 110 points, if your score exceeds 100, ***saturation arithmetic*** will be applied.
2) 可用中文作答

Section I: Multiple Choices

(If your answers are based on some assumptions you made, please write them down next to your answers so that you may earn partial credits even when the answers are incorrect)

PART A: (One correct answer only) (15 points)

1) **C** Which one of the following tools is NOT used in our compiler project?
   a) Lex/Flex
   b) Yacc/Bison
   c) Objdump/Readelf
   d) QEMU
   e) GVedit
   f) Make/Pmake

2) **AB** Which one of the following RISC-V registers is considered a caller-save register?
   a) X10
   b) X1
   c) X9
   d) X18
   e) X8

3) **D** In the following structure declaration in C, what would be **sizeof(struct S)**, if the program is compiled for RISC-V?

```
struct S {
   short a;
   long long b;
   char d;
```

```
    short e;
}
```

a) 13 bytes
b) 19 bytes
c) 20 bytes
d) 24 bytes
e) 32 bytes

4) **E** If a C switch statement has ~100 cases, with **no *biased*** cases, which one of the following code should ***not*** be part of the generated code by the compiler?
   a) Jump Table
   b) Hashed search table
   c) Binary search table
   d) Linear search table
   e) Cascaded IF's

5) **D** When compiling a language like Java, which one of the following semantic analyses is not required for the **IF** construct?
   a) The control expression must return a Boolean value.
   b) Whether each component is reachable
   c) Whether each component terminates normally
   d) Whether the variables are uninitialized
   e) Whether exception handling follows the ***catch or declare*** rule.

6) **C** Which one of the following is effective in detecting CSE (Common Sub-Expression)?

   a) Belady's algorithm
   b) Urshov labeling algorithm
   c) Value numbering
   d) Graph coloring
   e) A DAG algorithm

7) **B** Most compiler analyses are conservative. For example, in the reachability analysis, when we say a statement is reachable, we actually mean the statement ***may be*** reachable. Which one of the following analyses is ***definitive*** rather than conservative?

   a) A loop will terminate normally
   b) A variable has not been declared before use
   c) A variable is initialized
   d) An exception will be thrown in a try block
   e) An interference graph is not R-colorable using Greg Chaitin's algorithm

8) **D** Why is our C— compiler considered a multi-pass compiler?
   a) It can scan the program multiple times

b) It can build the symbol table multiple times
c) It can parse the program multiple times
d) It can walk the abstract syntax tree multiple times
e) It can conduct multiple different optimization passes

9) **C** Array dimension sizes are usually stored in the symbol table. However, for **dynamic arrays**, where the dimensional sizes are **NOT** known at compile time, where should the dimensional information be stored when they became known?

a) Store in the heap
b) Store in the cloud
c) Store in the dope vector allocated on the frame
d) Store in a global variable
e) Store in the dynamic library linkage table

10) **C** Which one of the following semantic checks is not needed in our C— compiler?
a) Every variable must be defined explicitly
b) A call to a function must use the correct number of parameters
c) Array indices must between bounds
d) A variable cannot be declared more than once
e) An array cannot be assigned to a scalar variable

11) **A** Intel has recently been troubled by a security "**bug**" (i.e. Meltdown and Spectre Attacks) in processors which is related to speculative execution. Which one of the following instructions is speculative by nature?
a) Conditional branch instruction
b) Data cache prefetching instruction
c) Break instruction
d) Procedure return instruction
e) An indirect jump

12) **B** Which one of the following textbooks has extensive discussions on ILP (Instruction Level Parallelism) related compiler optimizations?
a) Crafting a Compiler
b) The Dragon book (Chapter 10)
c) The Tiger book
d) Building an Optimizing Compiler
e) Engineering a Compiler

13) **C** In C, non-local names can be accessed efficiently. However, in some PLs, non-local names are accessed via static chains. This is because such languages allow
a) Nested If's
b) Nested Loops
c) Nested Procedures

d) Nested Structures
e) Nested Switches

14) **B** Google Chrome 55 (2016) is built with PBO optimizations, its start-up time becomes 17% faster. Which of the following contributes most to the performance gain?

a) Profile guided register allocation
b) Instruction cache prefetching
c) Profile guided code layout
d) Data cache prefetching
e) Procedure Inlining

15) **D** Which one of the following statements is NOT true for our C-- compiler project?
a) Our compiler generates RISC-V assembly code
b) Our compiler pays no attention to register usage
c) Our compiler has five check points: scan, parse, semantic check, code gen1 and code gen 2.
d) Our compiler handles non-local names

PART B (One or more correct answers) (30 points)

16) **BCD** Which of the following procedures **do not** need AR (Activation Record)?
a. Recursive procedures
b. Leaf procedures
c. Non-recursive procedures
d. Procedures do not involved in a recursive cycle of calls

17) **AC** Why do we generate RISC-V assembly code rather than VISA (Virtual ISA) code in our compiler project?
a) So that you are exposed to register assignment issues
b) So that you understand how to translate control constructs
c) So that you understand the overhead of procedure prologue and epilogue
d) So that you understand how exceptions are handled

18) **ABCD** Why most PLs adopt name equivalence rather than structure equivalence for type checking?
a. Implementation is easier – type comparison is just a simple comparison of two type pointers of descriptors.
b. Programmers could get full benefit of data types
c. It is very time consuming to tell whether two structures are structurally different
d. Type-inferred aliasing could be more effective

19) **BCD** Having a call graph can be very useful for compiler optimizations. However, building a call graph is often hindered by which of the following programming/compilation practices?

   a. Recursive calls
   b. Indirect calls
   c. Separate compilation
   d. Dynamic linking
   e. Pure functions

20) **ACE** Consider the following production rule in CFG, for example,

expr : expr + term

What could be the attributes for the non-terminal **expr** ?

a) Type
b) Storage location, e.g., offset to the frame pointer
c) Register or temporary used to hold the value
d) Value (in case of constant expressions)
e) Index value when value numbering is used to detect CSE

21) **BCD** Which of the following are the advantages of IR (Intermediate Representation)?

   a. Compile time can be reduced
   b. Machine independent optimizations can be shared by multiple back-ends
   c. Using IR support multiple-pass compilation
   d. Support multiple languages and ISAs

22) **ABE** Why should a compiler change the order of generated instructions?

a) In order to use fewer registers than the original order (Reorder can reduce a register' s lifetime.)
b) In order to minimize stalls in pipelined execution
c) In order to improve the reliability of the code
d) In order to speed up the compilation time
e) In order to decrease the code size (using fewer register may reduce load/store instruction)

23) **ABC** Which of the following are true about type checking?

   a. C Compilers use name equivalence for structured types
   b. C Compilers use structure equivalence for arrays
   c. Most modern PLs use name equivalence for type checking
   d. All type checking can be done at compile time
   e. The main purpose of type checking is to reduce program bugs

24) **ABC** Which of the following trends of PL and compilation are true?

   a. More popular languages are script languages
   b. More languages require interpretation
   c. More languages can benefit from dynamic compilation

d. More languages are pure statically compiled

25) **ABCD** To conduct graph coloring register allocation, which of the following steps are required?

a. Allocating variables, temporaries, or CSE's to unlimited pseudo registers
b. Using data flow analysis to determine pseudo register lifetime
c. Building interference graph
d. Check if the interference graph is R-colorable
e. If the graph is not R-colorable, then quit

26) **ABCE** Which of the following statements about Value numbering are true?

a) Value numbering can be used to build DAGs
b) Value numbering is used to capture local CSE
c) Value number means the index to an array storing CSE information
d) Using value numbering will likely increase the number of registers used
e) Using value numbering will likely decrease the number of instructions generated (Using fewer registers means less load/store instructions)

27) **ACE** Why does a compiler build a DAG instead of an expression tree for each basic block?

a) To capture CSE (Common Sub-expressions)
b) To obtain better Sethi-Ullman numbers for register allocation
c) To support instruction scheduling
d) To enable automatic instruction selections
e) To assist graph-coloring register allocation

28) **ACD** Which of the following statements about code generation of control structures are true?

a) In order to handle nested control structures, the label number of each control structure must be kept on a stack
b) The label number can be store on the value stack (i.e. semantic stack) in one-pass compilation
c) The label number can be stored on the runtime stack as a local variable in the code generation routines when recursive code generation is used.
d) The label number of each control structure must be unique
e) The label number should be stored in a global variable

29) **AC** If we want to minimize code size, which of the following optimizations should be turned off?

a) Loop unrolling
b) Procedure inlining
c) Function and loop alignments
d) DCE (Dead Code Elimination)
e) Copy propagation

30) **ABD** When we generate code for array address calculation, we need to compute base_address + VP * size. For the VP*size, why do we generate ***slli*** instead of ***muli*** ?

a) In our compiler project, size is always 4
b) ***SLLI*** instruction has a shorter latency
c) ***MULI*** instruction is a pseudo instruction, which will become two RV5 instructions, and take more code space.
d) A RISC-V processor may not implement the ***MULI*** instruction.

Section II: Short Essay Question **(65 points)**

1) Your Compiler Project (15 points)

What is the key to our multi-pass compiler implementation? **(1 point)**

maintain the AST tree for multipule pass

How do you recover from syntax errors in your C-- compiler? **(1 point)**

by setting g_anyErrorOccur bit to 1 and print error message.

How do you build your symbol table to handle the scope rules? **(1 points)**

Using many small tables to maintain each scope of table.

Our compiler allows arguments to be passed through a0 to a7 registers. However, during a function call, the caller still pushes stack space for the number of arguments. Why? **(2 points)**

The number of parameters may be over 8.

Our compiler uses both **sp** and **fp** pointers. GCC allows users to skip the **fp** pointer by using the **-fomit-frame-pointer** option**.** Why do we choose to support both **fp** and **sp? (2 points)**

Easier to maintain AR size and storage address of local variables

Do you have to follow the RISC-V calling convention in your compiler code generation assignment? What if you do not follow the calling convention? What advantages you can take by not following the convension? **(3 points)**

No, but can get bonus if follow. Can use the registers more flexibly. No need to care about which the caller saved or callee saved registers are.

How do you divide up the compiler implementation work with your partner on the type checker assignment? **(3 points)**

I write the symboltable.c and write the semantic.c from bottom up.

How do you divide up the compiler implementation work with your partner on the code generation assignment? **(2 points)**

I do the declaration Node part and ID Node part.

2) **Leaf Routines** (**15 points**)

Leaf routines are important targets for compiler optimization.

Give one or more examples of leaf routines. (**2 point)**

standard math functions

Why leaf routines are considered more important than non-leaf routines? **(2 points)**

A set of important library routines(ex: sin(), cos()) can be implemented by leaf routine optimization, and leaf routines somtimes can be optimized by remove stack frames but using caller-saved register.

What compiler optimizations are usually applied to leaf routines and **why**? **(3 points)**

using caller saved register

without register saving overhead

What can be done to increase the number of leaf routines? **(2 point)**

Spilt large procedure into several small simple prcedures.

When applying procedure the in-lining optimization, should the compiler favor inlining leaf-routines or non-leaf routines? **(2 point)**

inlining leaf-rotines

A routine is called a **conditional** leaf routine if all procedure calls made in this routine are inside a rarely executed block. How do you optimize such **conditional** leaf routines? **(2 points)**

By moving the rarely executed block of instructions as a function and inline the original routine in order to improve instruction locality and save overhead for calling procedure.

3) **DAG (10 points)**

a) What is a DAG? What is the difference between a DAG and a tree? **(2 point)**

A directed graph with no directed cycles. There can be different ancestors link to same descendant(subtree) in DAG, while tree can' t.

b) In our compiler, we build syntax trees for expressions. How could we build a DAG instead of a tree? **(2 points)**

By value numbering, we can find out CSE and build DAG on indexes.

c) DAG is also used in code scheduling. What are the differences between a DAG for code scheduling and a DAG for code generation? (**2 point**)

DAG for code scheduling is to measure the dependency between instructions. While DAG for code generation is to minimize the register use and number of instructions.

d) How do we figure out what would be the minimum number of registers required for an expression tree? **(2 points)** How about for a DAG? **(2 points)**

Tree: Using Sethi-Ullman/Ershov Numbering to find minimum number of registers use in expression tree.

DAG: NP-hard, We can use enumeration algorithm (which enumerate all possible answers to check if it' s valid) in O(n!) , or using dynamic programming scheme and reordering techniques in O(n2^(2n)) which publish in this paper: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.40.741&rep=rep1&type=pdf

4) Activation record (AR) (10 points)

An Activation Record (AR) is a data area associated with a procedure, it is pushed onto a runtime stack when the procedure is called, and popped off when the procedure returns.

List at least three **optimizations** that could lead to a smaller AR. (**3 points**)

Using calling convention
Procedure In-Lining
leaf routine optimization

Some procedures do not need an AR to be created. What are such procedures? How does a compiler identify them? (**2 points**)

Non-recursive procedure, Leaf procedure.
By building a call graph

Do we need to ensure ARs are always 8 byte aligned? If we do, explain why. (**3 points**)

Yes, If we don't align the bytes, data may be placed on different pages of memory, which lead to high accessing cost.

In our C-- compiler, we only support **Int** and **Float** data types, so ARs should be 4 byte aligned, why in (c) we said 8-byte aligned? List the objects on the ARs that are 8 bytes. (**2 points**)

Access link
Cntrol link

5) Register Allocation (10 points)

Register allocation has long been known as an NP-hard problem.

How does the problem get simplified to avoid being NP-hard? **(3 points)**

Reduce the question “What is the minimun number of colors required for coloring the graph” to “Is the graph R-colorable”, then use graph pruning to find the approximate answer.

The Intel/HP Itanium processor has 128 general purpose registers and 128 floating point registers. What issues will the compiler face if the graph coloring algorithm is adopted for register allocation? How to fix or avoid the problems? **(4 points)**

Procedure call and save cost will be too high.Instead of asking “Is the graph 128-colorable”, try asking “Is the graph 32-colorable” or “Is the graph 16-colorable” to reduce the cost.

However, even when we reduced the difficult problems into linear time complexity, register allocation is still one of the most time consuming part in the compiler. Explain why! **(3 points)**

Building the interference graph is still highly costed. For example, if there is 100 variables in a basic block, each of them has to compare to others if their lifetime is overlap or not.

6) **Explaining** the following acronyms in the context of compilers. (**5 points**)

TVM

A machine learning compiler framework for CPUs, GPUs, and machine learning accelerators.

AOT

AOT compilation is one way of improving the performance of Java programs and in particular the startup time of the JVM.

LTO

Link Time Optimization, which gives GCC the capability of dumping its internal representation to disk, so that all the different compilation units that make up a single executable can be optimized as a single module.

MLIR

MLIR aims to address software fragmentation, improve compilation for heterogeneous hardware, significantly reduce the cost of building domain specific compilers, and aid in connecting existing compilers together.

GPUCC

An open-source GPGPU compiler, which based on LLVM and supports C++11 and C++14.