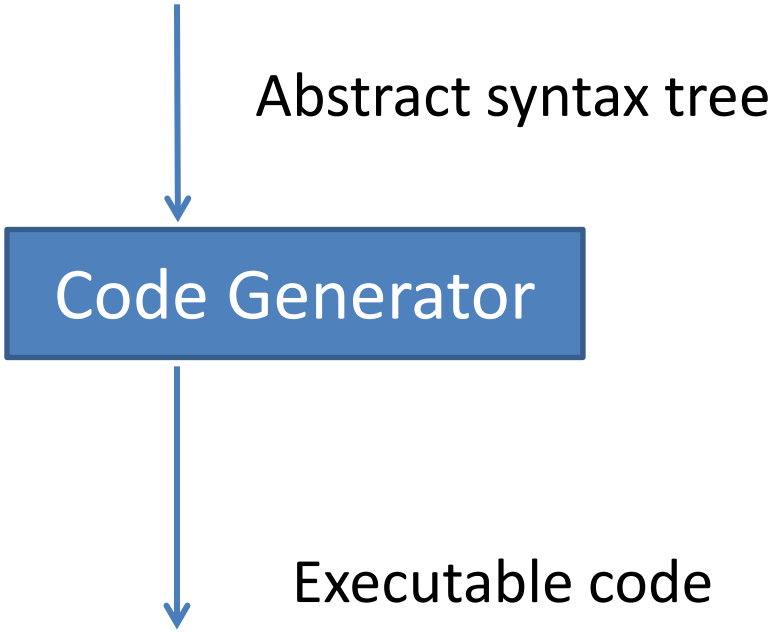
# CZ3007 Compiler Techniques

## Review (Week 9)

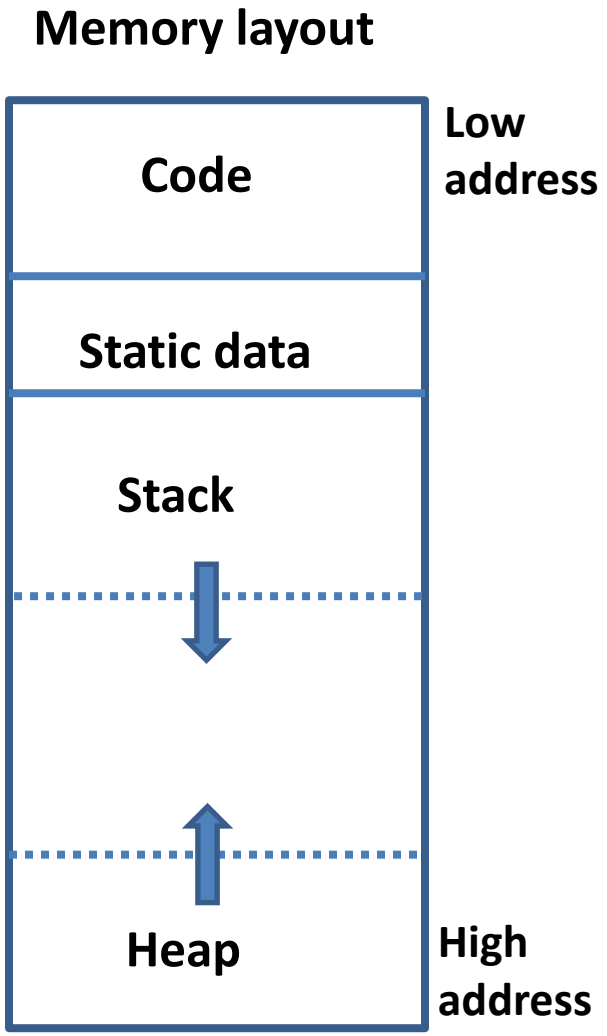
# Recap

1. Introduction/Overview
   1) What a code generator will do
   2) Runtime environment of a program
   3) Multi-pass process of generating and optimising code
   4) Interpreting VM code and just-in-time compilation
2. Overview of the JVM (up to slide 36)
   1) Runtime memory organisation
   2) Bytecode
   3) Stack-based computation

# What a code generator will do




# Runtime environment of a program



# Multi-pass process of generating and optimising code




# Interpreting VM code and just-in-time compilation

# Poll 1

In the runtime memory layout of a program (in a language like C, C++)

1. The code area is of fixed size or variable size?
2. The static data area is of fixed size or variable size?
3. The stack area is of fixed size or variable size?
4. The heap area is of fixed size or variable size?

# Poll 1

In the runtime memory layout of a program (in a language like C, C++)

1. The code area is of fixed size or variable size? **Fixed**
2. The static data area is of fixed size or variable size? **Fixed**
3. The stack area is of fixed size or variable size? **Variable**
4. The heap area is of fixed size or variable size? **Variable**

# Poll 2

Is it possible that running code on a virtual machine like the Java Virtual Machine could be faster than running native code?

# Poll 2

Is it possible that running code on a virtual machine like the Java Virtual Machine could be faster than running native code?

- **Optimization done by a compiler is based on static information. E.g.**
  - **avoid re-computation of some expressions.**
  - **x = x^2; ⇒ x = x*x;**
  - **x = 5*2*y; ⇒ x = 10*y;**
- **JVM can do more targeted optimization so it could be faster in some cases. E.g.**
  - **An example of more targeted optimization is that the JIT compilation may be able to use specific CPU model's features not generally available to achieve faster execution (like vector instructions)**
  - **Many optimizations are only feasible at runtime**

# Recap

1. Introduction/Overview
   1) What a code generator will do
   2) Runtime environment of a program
   3) Multi-pass process of generating and optimising code
   4) Interpreting VM code and just-in-time compilation
2. Overview of the JVM (up to slide 36)
   1) Runtime memory organisation
   2) Bytecode
   3) Stack-based computation

# Poll 3

In the runtime memory organization of a Java program

1. Is the method area one per thread or shared?
2. Is the heap area one per thread or shared?
3. Is the stack area one per thread or shared?

# Poll 3

In the runtime memory organization of a Java program

1. Is the method area one per thread or shared? **Shared.**
2. Is the heap area one per thread or shared? **Shared.**
3. Is the stack area one per thread or shared? **One per thread.**

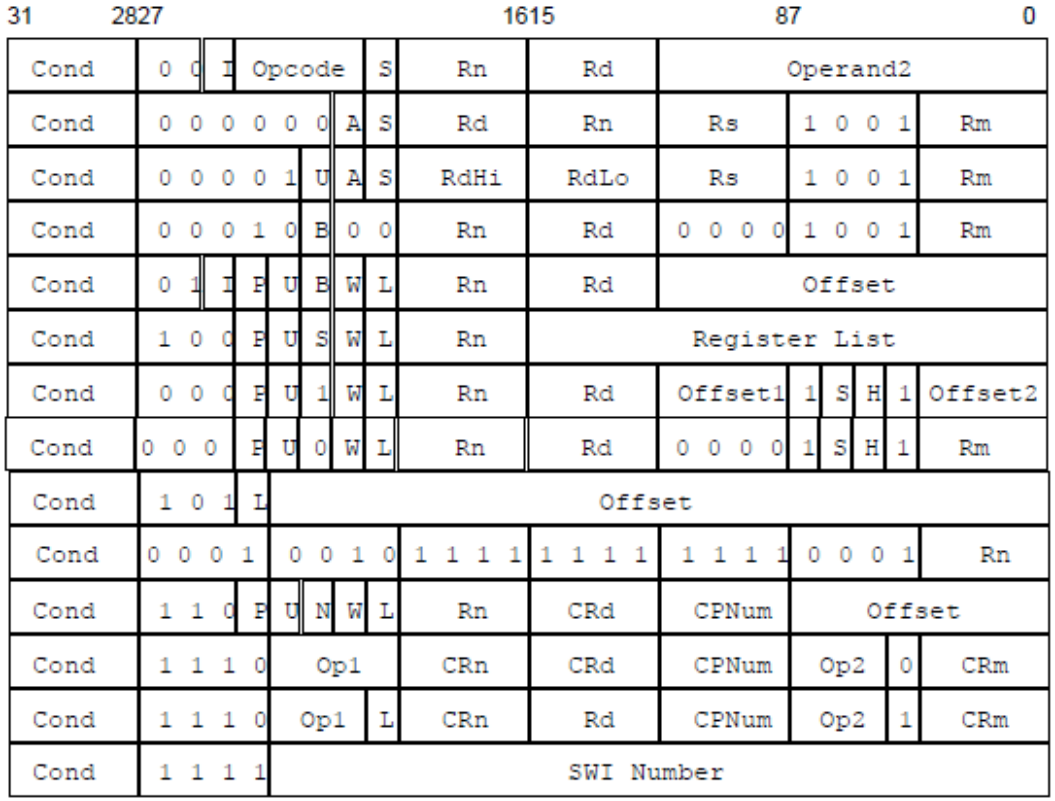
# Question

- Using JVM bytecode as an example.  Why do we say that bytecode is often more compact than native code?

back

# ARM Instruction Set Format

Advanced RISC Machines




**Instruction type**

- Data processing / PSR Transfer
- Multiply
- Long Multiply **(v3M / v4 only)**
- Swap
- Load/Store Byte/Word
- Load/Store Multiple
- Halfword transfer : Immediate offset (v4 only)
- Halfword transfer: Register offset (v4 only)
- Branch
- Branch Exchange **(v4T only)**
- Coprocessor data transfer
- Coprocessor data operation
- Coprocessor register transfer
- Software interrupt

Please complete the online teaching feedback by 11.59pm, this Sunday.

*Thank you!*