EECS 583 – Advanced Compilers
Course Overview, Introduction to Control Flow Analysis

Winter 2024, University of Michigan

Jan 10, 2024

https://web.eecs.umich.edu/~mahlke/courses/583w24

Lecture + Discussion

- Class meeting pattern
  - Lecture: Mon/Wed 10:30-12:00
    - Scott office hours right after lecture: noon-12:30
  - Discussion: Wed 12:30-1:30, 1005 Dow (optional, some cancelled)
    - Homework details, questions
- In-person lecture: G906 Cooley (White Auditorium)
  - Try to attend, get more out of the lecture
  - Please stay home if you are sick!
  - May have some virtual classes during the semester
- Lecture will also be presented live on Zoom
  - Participate from home if you wish
  - Zoom videos also available (try to use just for review!)
    - Bad idea if this is all you do: run 1.5x, multi-task, don't pay attention
  - Discussion section is not recorded
- Zoom info
  - Same link/password for all lectures, posted on course website
  - Separate link for GSI office hrs – more later

Attending Class Virtually

- Lecture is synchronous and recorded
  - Please try to attend live if you can
  - We'll start at 10:35 sharp
  - **Keep your camera and mic muted**
    - **Critical to avoid disruptions**
- Asking questions on Zoom
  - Type the word "question" in the chat box
  - GSI will unmute you and you can ask question
  - If you prefer not to speak, then just type out your question in chat and the GSI can ask it for you
  - I will also pause regularly to ask if there are questions
  - Discussion important in a grad class, so don't be bashful

About Me

- Mahlke = mall key
  - But just call me Scott
- Been at Michigan since 2001
  - Compiler guy who likes hardware
  - Program optimization to make programs go faster
  - Building custom hardware for high performance/low power
- Before this – HP Labs in Silicon Valley
- Before before – Grad student at UIUC
- Before ^ 3 – Undergrad at UIUC

More About Me

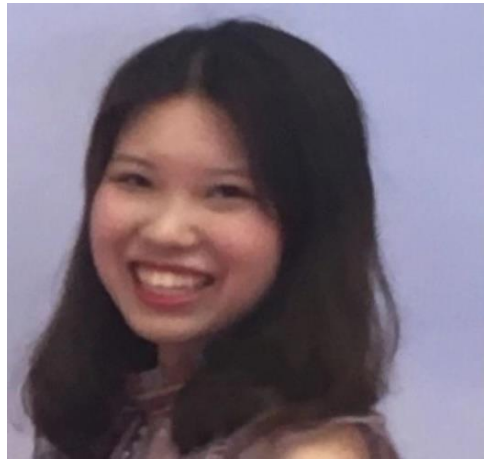
- 3 kids – 7, 7, and 5
  - So if I show up to lecture half asleep, you know why!

Contact Information

- Email: mahlke@umich.edu
- Office hours
  - Mon/Wed 12:00-12:30 right outside lecture room
  - Or send me an email for an appointment
- Visiting office hrs
  - Mainly help on classroom material, concepts, etc.
  - I am an LLVM novice, so likely I cannot answer any non-trivial question
  - See GSIs for LLVM details

583 GSIs

- Yunjie Pan (panyj@umich.edu)
  - Office hours: Mon 1-3pm, Tue 1-3pm, Wed 1:30-3:30pm
- Aditya Vasudevan (adivasu@umich.edu)
  - Office hours: Tue 3-5pm, Thu 3-5pm, Fri 10am-12pm
- Location: Zoom (link on course website, same link for the entire semester, same for both GSIs, passcode = eecs583)



Getting Help from the GSIs

- LLVM help/questions
- But, you will have to be independent in this class
  - Read the documentation and look at the code
  - Come to them when you are really stuck or confused
  - They cannot and will not debug your code
  - Helping each other is encouraged
  - Use the class piazza group (GSIs will monitor)
- Virtual office hours on Zoom
  - Will offer a combination of appointments (EECS 583 calendar) and open slots. Details posted on piazza.

Class Overview

- This class is NOT about:
  - Programming languages
  - Parsing, syntax checking, semantic analysis
  - Handling advanced language features – virtual functions, …
  - Frontend transformations
  - Debugging
  - Simulation
- Compiler backend
  - Mapping applications to processor hardware
  - Retargetability – work for multiple platforms (not hard coded)
  - Work at the assembly-code level (but processor independent)
  - Speed/Efficiency
    - How to make the application run fast
    - Use less memory (text, data), efficiently execute
    - Parallelize, prefetch, optimize using profile information

Background You Should Have

- 1. Programming
  - Good C++ programmer (essential)
  - Linux, gcc, emacs (vi or other editor is ok too)
  - Debugging experience – hard to debug with printf's alone – gdb!
  - Compiler system not ported to Windows
- 2. Computer architecture
  - EECS 370 is good, 470 is better but not essential
  - Basics – caches, pipelining, function units, registers, virtual memory, branches, multiple cores, assembly code
- 3. Compilers
  - Frontend stuff is not very relevant for this class, but good to know
  - Basic backend stuff we will go over fast
    - Non-EECS 483 people will have to do some supplemental reading

Textbook and Other Classroom Material

- No required text – Lecture notes, papers
- LLVM compiler system – we will use version 17.0.6
  - LLVM webpage: http://www.llvm.org
  - Read the documentation!
  - LLVM users group
- Course webpage + course newsgroup
  - https://www.eecs.umich.edu/~mahlke/courses/583w24
  - Lecture notes – available the night before class
  - Piazza – ask/answer questions, GSIs and I will try to check regularly but may not be able to do so always
    - http://www.piazza.com

What the Class Will be Like

- Core backend stuff
  - Text book material – some overlap with 483
  - 2 homeworks to apply classroom material
- Research papers
  - Last 1/3rd of the semester, students take over
  - Select paper related to your project
  - Each project team - presents 1 paper. 15 min talk + Q&A.
  - Entire class is expected to watch presentations and grade presentations
  - You will need to attend live for at least your own presentation

What the Class Will be Like (2)

- Learning compilers
  - No memorizing definitions, terms, formulas, algorithms, etc
  - Learn by doing – Writing code
  - Substantial amount of programming
    - Fair learning curve for LLVM compiler
  - Reasonable amount of reading
- Classroom
  - Attendance – Best to join live, lots of examples solved in class
  - Discussion important
    - Work out examples, discuss papers, etc
  - Essential to stay caught up
  - Extra meetings outside of class to discuss projects

Course Grading

- Yes, everyone will get a grade
  - Grad class: Most (hopefully all) will get A's and B's
  - Poor grades on homeworks are big problem
- Components
  - Midterm exam – 25%
  - Project – 45%
  - Homeworks – 15%
  - Paper presentation – 10%
  - Class participation – 5%

Homeworks

- 1 preliminary (HW0), available on course webpage now
  - Get LLVM set up, nothing to submit
- 2 real homeworks
  - 1 small &1 harder programming assignment
  - Design and implement something we discussed in class
- Goals
  - Learn the important concepts
  - Learn the compiler infrastructure so you can do the project
- Grading
  - Working testcases?, Does anything work? Level of effort?
- Working together on the concepts is fine
  - Make sure you understand things or it will come back to bite you
  - Everyone must do and turn in their own assignment

Projects – Most Important Part of the Class

- Design and implement an "interesting" compiler technique and demonstrate its usefulness using LLVM
- Topic/scope/work
  - 3-5 people per project (Other group sizes allowed in some cases)
  - You will pick the topics (I have to agree)
  - You will have to
    - Read background material
    - Plan and design
    - Implement and debug
- Deliverables
  - Working implementation
  - Project report: ~5 page paper describing what you did/results
  - 15 min presentation at end (demo if you want)
  - Project proposal (early March) scheduled with each group during semester

Types of Projects

- New idea
  - Small research idea
  - Design and implement it, see how it works
- Extend existing idea
  - Take an existing paper, implement their technique
  - Then, extend it to do something small but interesting
    - Generalize strategy, make more efficient/effective
- Implementation
  - Take existing idea, create quality implementation in LLVM
  - Try to get your code released into main LLVM system
- Using other compilers/systems (GPUs, JIT, mobile phone, etc.) is possible

Topic Areas (You are Welcome to Propose Others)

- Automatic parallelization
  - Loop parallelization
  - Vectorization/SIMDization
  - Transactional memories/speculation
  - Breaking dependences
- Memory system performance
  - Instruction/data prefetching
  - Use of scratchpad memories
  - Data layout
- Reliability
  - Catching transient faults
  - Reducing AVF
- Customized hardware
  - High level synthesis
  - HW optimization
- Power
  - Instruction scheduling techniques to reduce power
  - Identification of narrow computations
- Streaming/GPUs
  - Stream scheduling
  - Memory management
  - Optimizing CUDA programs
- Security
  - Program analysis to identify vulnerabilities
  - Eliminate vulnerabilities via xforms
- Dynamic optimization
  - DynamoRIO
  - Run-time optimization

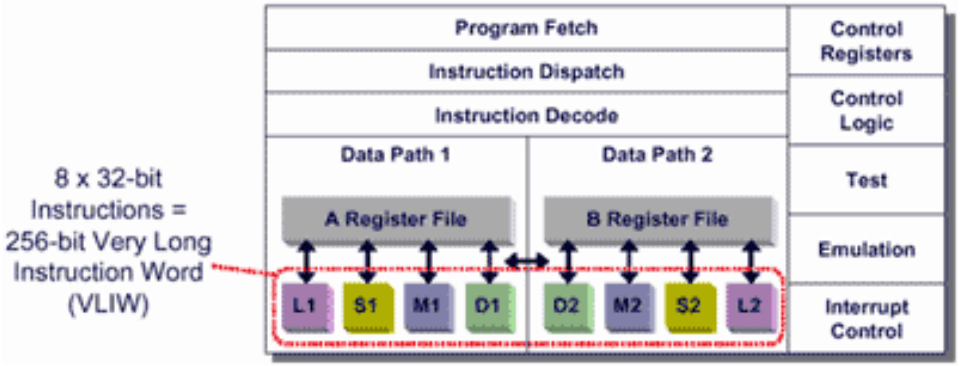
Class Participation

- Interaction and discussion is essential in a graduate class
  - Try to join live if you can (not required)
  - If you are here, don't just stare at the wall
  - Be prepared to discuss the material
  - Have something useful to contribute
- Opportunities for participation
  - Research paper presentations – thoughts, comments, questions
  - Saying what you think during class or in project discussions outside of class
  - Lectures: Solving class problems, asking questions
  - Helping answer questions on piazza!

Class Schedule (on course website)

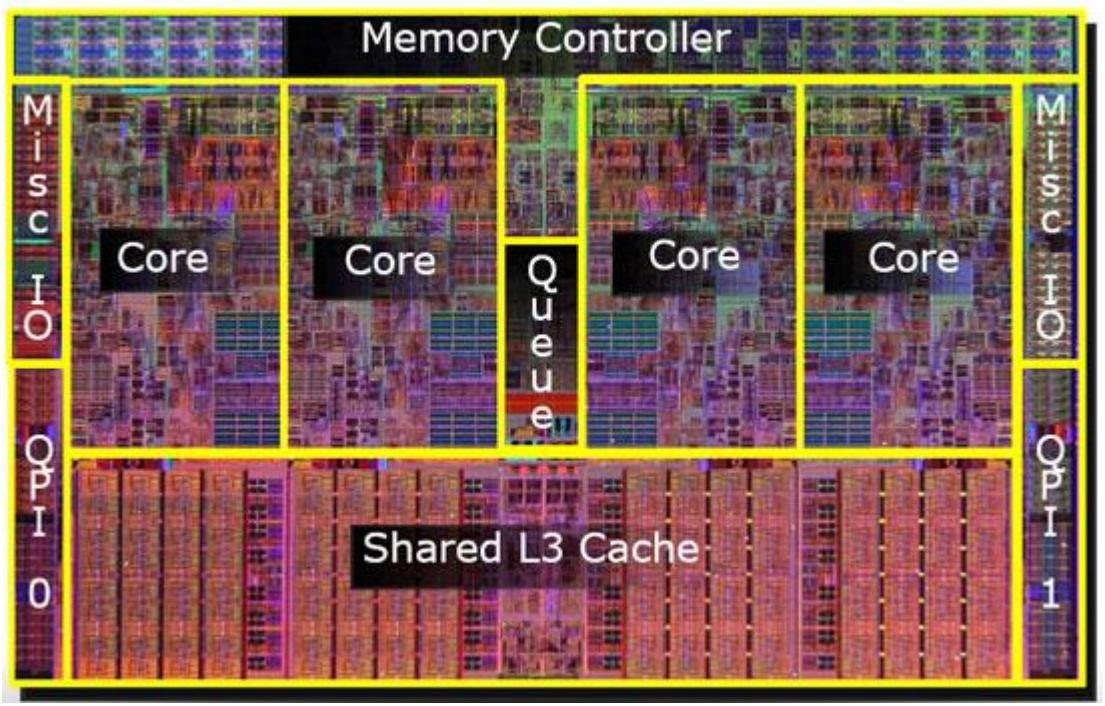
Week	Date	Topic
1 Mon	-	
Wed	Jan 10	Course intro, Control flow analysis, HW #0 out
2	Jan 15	No class, MLK Day
	Jan 17	Control flow analysis, HW #0 due (nothing to turn in), HW #1 out
3	Jan 22	Control flow analysis
	Jan 24	Control flow analysis
4	Jan 29	Dataflow analysis, HW #1 due
	Jan 31	Dataflow analysis, HW #2 out
5	Feb 5	SSA form
	Feb 7	Code optimization
6	Feb 12	Code optimization
	Feb 14	Code generation
7	Feb 19	Code generation
	Feb 21	Code generation, HW #2 due
8	Feb 26	No class, Spring Break
	Feb 28	No class, Spring Break
9	Mar 4	Code generation
	Mar 6	Code generation
10	Mar 11	No regular class - Project proposals
	Mar 13	No regular class - Project proposals
11	Mar 18	Midterm Review
	Mar 20	Midterm Exam
12	Mar 25	Research paper presentations
	Mar 27	Research paper presentations
13	Apr 1	Research paper presentations
	Apr 3	Research paper presentations
14	Apr 8	Research paper presentations
	Apr 10	Research paper presentations
15	Apr 15	Research paper presentations
	Apr 17	Research paper presentations
16	Apr 22	No regular class – Finish projects
	-	
	Apr 23-29	Project demos

Target Processors: 1) VLIW/EPIC Architectures



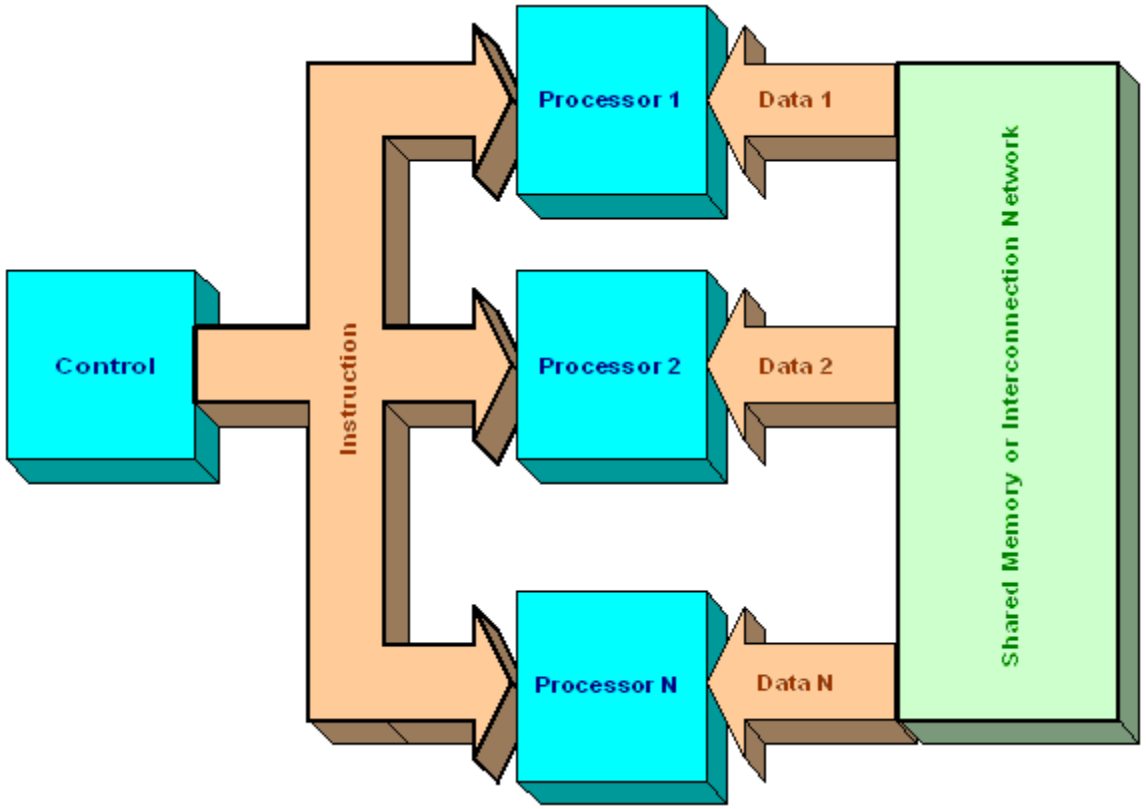
- VLIW = Very Long Instruction Word
  - Aka EPIC = Explicitly Parallel Instruction Computing
  - Compiler managed multi-issue processor
- Desktop
  - IA-64: aka Itanium I and II, Merced, McKinley
- Embedded processors
  - All high-performance DSPs are VLIW
    - Why? Cost/power of superscalar, more scalability
  - TI-C6x, Philips Trimedia, Starcore, ST-200

Target Processors: 2) Multicore



- Sequential programs – 1 core busy, 3 sit idle
- How do we speed up sequential applications?
  - Switch from ILP to TLP as major source of performance
  - Memory dependence analysis becomes critical

Target Processors: 3) SIMD/GPU



- Do the same work on different data: GPU, SSE, etc.
- Energy-efficient way to scale performance
- Must find “vector parallelism”

So, lets get started… Compiler Backend IR – Our Input

- Variable home location
  - Frontend – every variable in memory
  - Backend – maximal but safe register promotion
    - All temporaries put into registers
    - All local scalars put into registers, except those accessed via &
    - All globals, local arrays/structs, unpromotable local scalars put in memory. Accessed via load/store.
- Backend IR (intermediate representation)
  - machine independent assembly code – really resource indep!
  - aka RTL (register transfer language), 3-address code
  - r1 = r2 + r3 or equivalently add r1, r2, r3
    - Opcode (add, sub, load, …)
    - Operands
      - Virtual registers – infinite number of these
      - Literals – compile-time constants

First Topic: Control Flow Analysis

- Control transfer = branch (taken or fall-through)
- Control flow
  - Branching behavior of an application
  - What sequences of instructions can be executed
- Execution → Dynamic control flow
  - Direction of a particular instance of a branch
  - Predict, speculate, squash, etc.
- Compiler → Static control flow
  - Not executing the program
  - Input not known, so what could happen
- Control flow analysis
  - Determining properties of the program branch structure
  - Determining instruction execution properties

Basic Block (BB)

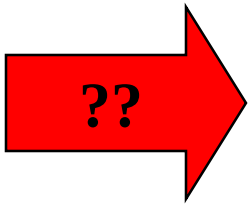
- Group operations into units with equivalent execution conditions

- Defn: Basic block – a sequence of consecutive operations in which flow of control enters at the beginning and leaves at the end without halt or possibility of branching except at the end
  - Straight-line sequence of instructions
  - If one operation is executed in a BB, they all are

- Finding BB's
  - The first operation in a function starts a BB
  - Any operation that is the target of a branch starts a BB
  - Any operation that immediately follows a branch starts a BB

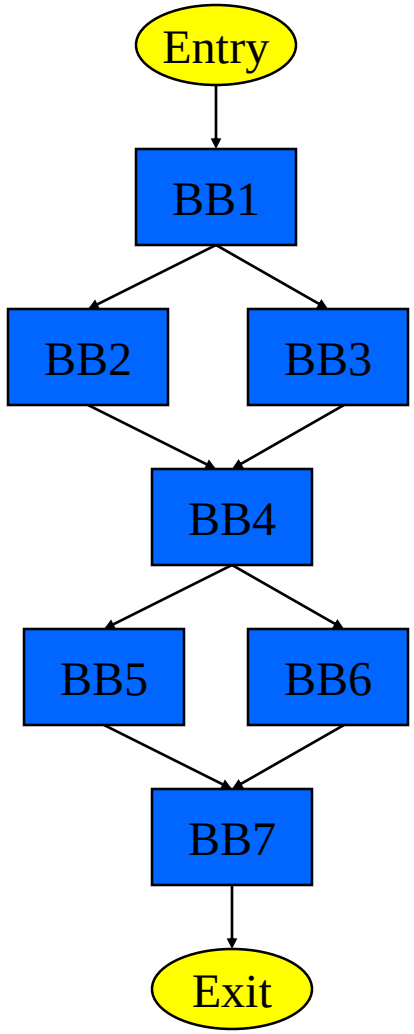
Identifying BBs - Example

L1: r7 = load(r8)
L2: r1 = r2 + r3
L3: beq r1, 0, L10
L4: r4 = r5 * r6
L5: r1 = r1 + 1
L6: beq r1 100 L3
L7: beq r2 100 L10
L8: r5 = r9 + 1
L9: jump L2
L10: r9 = load (r3)
L11: store(r9, r1)



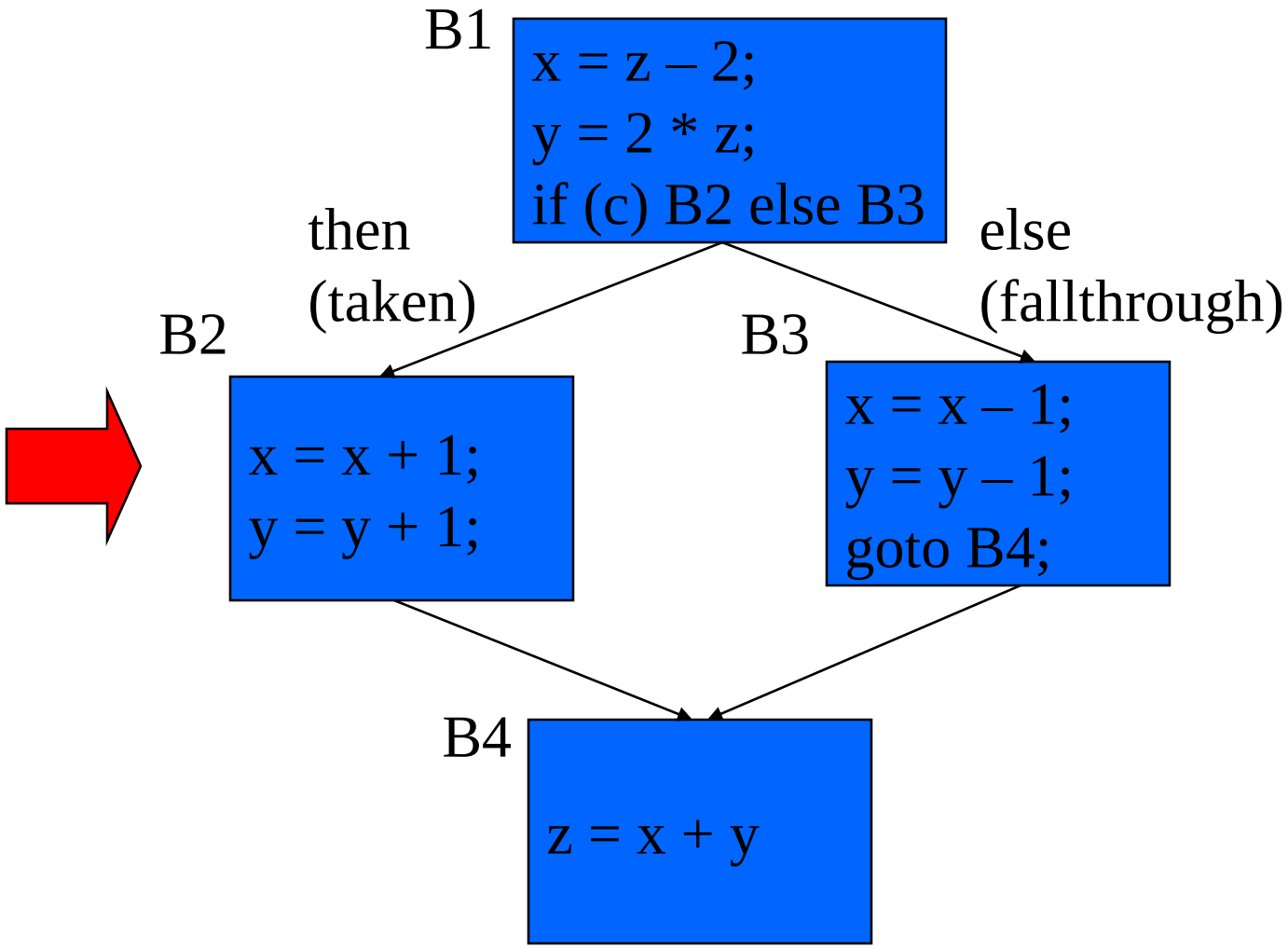
Control Flow Graph (CFG)

- Defn Control Flow Graph – Directed graph, G = (V,E) where each vertex V is a basic block and there is an edge E, v1 (BB1) → v2 (BB2) if BB2 can immediately follow BB1 in some execution sequence
  - A BB has an edge to all blocks it can branch to
  - Standard representation used by many compilers
  - Often have 2 pseudo vertices
    - entry node
    - exit node



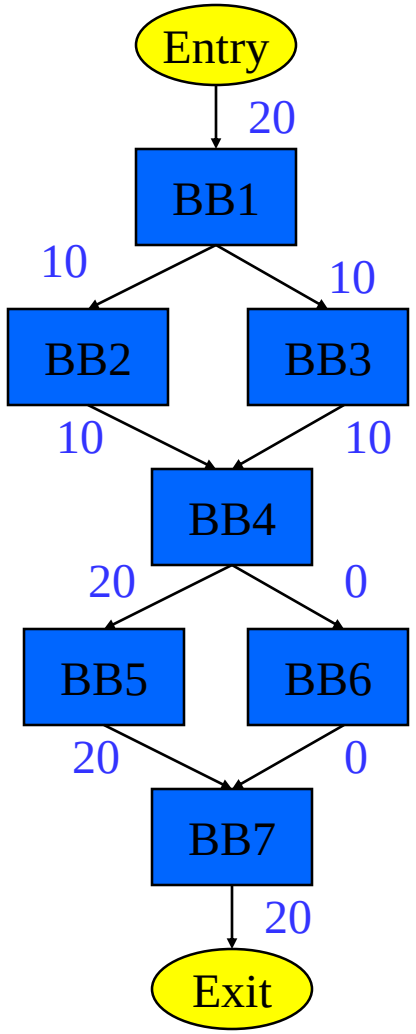
CFG Example

```
x = z – 2;
y = 2 * z;
if (c) {
    x = x + 1;
    y = y + 1;
}
else {
    x = x – 1;
    y = y – 1;
}
z = x + y
```



Weighted CFG

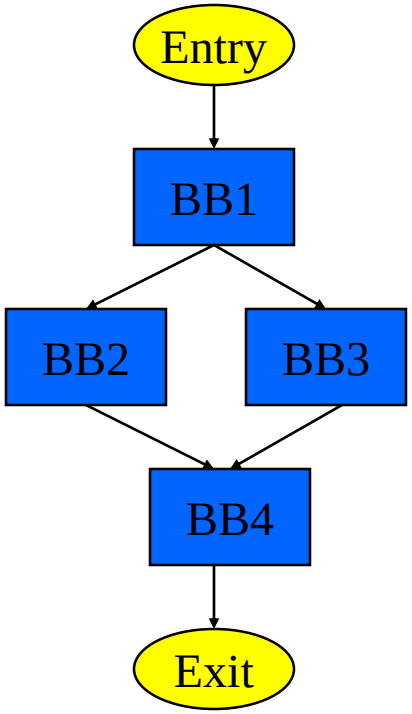
- Profiling – Run the application on 1 or more sample inputs, record some behavior
  - » Control flow profiling
    - edge profile
    - block profile
  - » Path profiling
  - » Cache profiling
  - » Memory dependence profiling
- Annotate control flow profile onto a CFG → weighted CFG
- Optimize more effectively with profile info!!
  - » Optimize for the common case
  - » Make educated guess



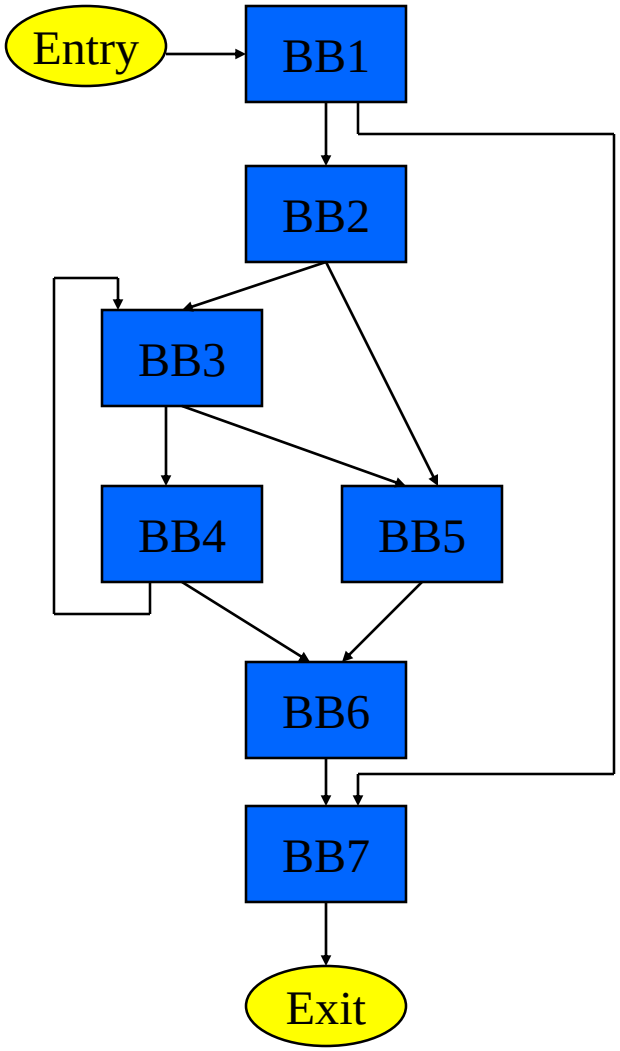
Property of CFGs: Dominator (DOM)

- Defn: Dominator – Given a CFG(V, E, Entry, Exit), a node x dominates a node y, if every path from the Entry block to y contains x
- 3 properties of dominators
  - Each BB dominates itself
  - If x dominates y, and y dominates z, then x dominates z
  - If x dominates z and y dominates z, then either x dominates y or y dominates x
- Intuition
  - Given some BB, which blocks are guaranteed to have executed prior to executing the BB

Dominator Example 1



Dominator Example 2



Get Started ASAP!! Homework 0

- Go to http://llvm.org
- Setup LLVM 17.0.6 on the class server or your favorite Linux box
  - For server, use the central version that is already set up
  - For your own system, read the installation instructions
  - See Aditya's post on piazza for detailed instructions
- Try to run it on a simple C program
- HW1 goes out next week and you need LLVM
- We will have 2 dedicated servers for class use
  - eecs583a/eecs583b.eecs.umich.edu
  - Everyone should have ssh access