

## Plan for Today

### Arrays

- array assignment
- array access through array reference

### Switch Statements

#### Basic blocks

- tracing
- removing unnecessary jumps

#### Peephole optimizations

CS453 Lecture

Arrays and Control Flow

1

## Array Assign Example

```
class ArrayAssign {
    public static void main(String[] a){
        System.out.println(new MyClass().testing()); } }
class MyClass {
    int[] y;
    public int testing() {
        int[] x;
        y = new int [80]; x = new int [10];
        x[0] = 7;
        x[3] = 77;
        y[75] = 42;
        return x[0] + y[x[3]-2];
    }
}
```

CS453 Lecture

Arrays and Control Flow

2

## Compiling Control Flow

### Switch statements

- Convert **switch** into low-level IR

```
e.g., switch (c) {
    case 0: f();
           break;
    case 1: g();
           break;
    case 2: h();
           break;
}

if (c!=0) goto next1
call f,0
goto done
next1: if (c!=1) goto next2
call g,0
goto done
next2: if (c!=3) goto default
call h,0
goto done
default:
done:
```

- Optimizations (depending on size and density of cases)
  - Create a jump table (store branch targets in table)
  - Use binary search

CS453 Lecture

Arrays and Control Flow

3

## Basic blocks and traces

### Control flow

- what is the next instruction that can be executed after executing instr X?
- basic blocks are lists of statements with straight-line code control flow
- jumps and cjmps cause more interesting control flow

### Correctness

- the false body needs to be after each CJUMP because of fallthrough

### Optimization ( could actually do some of this in PA6 to remove jumps )

- if a target basic block follows the block that unconditionally jumps to it, then can remove the jump
- in a trace, a target basic block is placed after a block that jumps to it
- hot paths should be put in the same trace so as to avoid frequent pipeline stalls and icache misses

CS453 Lecture

Arrays and Control Flow

4

### Basic block and tracing example

---

```
f = new Foo();
y = new int [20];

if (this.otherFunc(y, f)) {
    local1 = 3;
    local2 = 4;
    local3 = 5 * local1 + y[77];
} else {
    local1 = 8;
    local2 = 9;
    local3 = 10 * local1;
}
return local3;
```

### Peephole Optimizations

---

#### What?

- transformations that target short sequences of code in basic blocks

#### Why?

- to make code generation easier, often the resulting code has inefficiencies
- goal is to do same work with fewer instructions