

CSE P 501 Su04 O-1

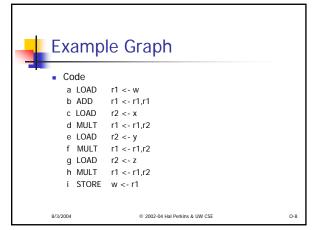


Precedence Graph

- Nodes n are operations
- Attributes of each node
 - type kind of operation
 - delay latency
- If node n2 uses the result of node n1, there is an edge e = (n1,n2) in the graph

8/3/2004

© 2002-04 Hal Porkins & LIW CSF





Schedules (1)

- A correct schedule S maps each node n into a non-negative integer representing its cycle number, and
 - S(n) >= 0 for all nodes n (obvious)
 - If (n1,n2) is an edge, then S(n1) + delay(n1) <= S(n2)
 - For each type t there are no more operations of type t in any cycle than the target machine can issue

8/3/2004

© 2002-04 Hal Perkins & UW CSE



Schedules (2)

The length of a schedule S, denoted L(S) is

$$L(S) = \max_{n} (S(\underline{n}) + delay(n))$$

- The goal is to find the shortest possible correct schedule
 - Other possible goals: minimize use of registers, power, space, ...

8/3/2004

© 2002-04 Hal Perkins & UW CSE

0-10



Constraints

- Main points
 - All operands must be available
 - Multiple operations can be ready at any given point
 - Moving operations can lengthen register lifetimes
 - Moving uses near definitions can shorten register lifetimes
 - Operations can have multiple predecessors
- Collectively this makes scheduling NP-complete
- Local scheduling is the simpler case
 - Straight-line code
 - Consistent, predictable latencies

© 2002-04 Hal Perkins & UW CSE



Algorithm Overview

- Build a precedence graph P
- Compute a *priority function* over the nodes in *P* (typical: longest latency-weighted path)
- Use list scheduling to construct a schedule, one cycle at a time
 - Use queue of operations that are ready
 - At each cycle
 - · Chose a ready operation and schedule it
 - Update ready queue
- Rename registers to avoid false dependencies and conflicts

© 2002-04 Hal Perkins & UW CSE

CSE P 501 Su04 O-2

```
List Scheduling Algorithm

Cycle = 1; Ready = leaves of P; Active = empty; while (Ready and/or Active are not empty)

if (Ready is not empty)

remove an op from Ready;

S(op) = Cycle;

Active = Active + op;

Cycle++;

for each op in Active

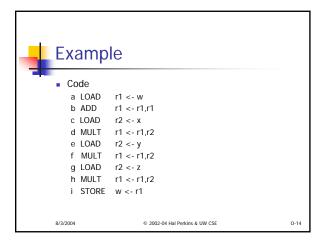
if (S(op) + delay(op) <= Cycle)

remove op from Active;

for each successor s of op in P

if (s is ready - i.e., all operands available)

add s to Ready
```





Variations

- Backward list scheduling
 - Work from the root to the leaves
 - Schedules instructions from end to beginning of the block
- In practice, try both and pick the result that minimizes costs
 - Little extra expense since the precedence graph and other information can be reused
- Global scheduling and loop scheduling
 - Extend basic idea in more aggressive compilers

8/3/2004

© 2002-04 Hal Perkins & UW CSE

CSE P 501 Su04 O-3