Introduction Scheduling (Part 1)
Introduction and Acyclic Scheduling

CS 380C: Advanced Compiler Techniques

Thursday, October 11th 2007
Lecture Overview

Code Generator
- Back end part of compiler (code generator)
- Instruction scheduling
- Register allocation

Instruction Scheduling
- Input: set of instructions
- Output: total order on that set
Lecture Overview

Introduction Constraints List Scheduling Conclusion

Lecture Outline

Lectures

1. Introduction and acyclic scheduling (today)
2. Software pipelining (Tuesday 23)

Today

- Definition of instruction scheduling
- Constraints
- Scheduling process
- Acyclic scheduling: *list scheduling*
Introduction to Instruction Scheduling

**Context**
- Backend part of the compiler chain (code generation)
- Inputs: set of instructions (assembly instructions)
- Outputs: a *schedule*
  - Set of scheduling dates (one date per instruction)
  - Total order

**Goal**
- Minimize the execution time (number of cycles)
- Different possible objective functions to minimize:
  - Power consumption
  - ...
Constraints

- Is it possible to generate any schedule?
Constraints

Is it possible to generate any schedule?

Example:

\[
\begin{align*}
  a &= b + c \\
  d &= a + 3 \\
  e &= f + d
\end{align*}
\]

Possibility to change instruction order?
Is it possible to generate any schedule?

Example:

\[
\begin{align*}
a &= b + c \\
d &= a + 3 \\
e &= f + d
\end{align*}
\]

Possibility to change instruction order?

No, because of data dependences

Flow dependences on \(a\) and \(d\)
 Constraints

- Data dependences enforce a partial order for the final schedule
- Other types of constraints?
Data dependences enforce a partial order for the final schedule

Other types of constraints?

Example:

\[
\begin{align*}
a &= b + c \\
d &= e + f \\
\end{align*}
\]

Target architecture with 1 ALU
Constraints

- Data dependences enforce a partial order for the final schedule
- Other types of constraints?

Example:

\[
\begin{align*}
  a &= b + c \\
  d &= e + f
\end{align*}
\]

- Target architecture with 1 ALU
- Impossible to use the same functional unit concurrently
- Resource constraints
Constraints

- Data dependences enforce a partial order for the final schedule
- Other types of constraints?

Example:

```
    a = b + c ;
    d = e + f ;
```

- Target architecture with 1 ALU
- Impossible to use the same functional unit concurrently
- Resource constraints

Two types of constraints: *data dependences* and *resource usage*
Constraints influencing Instruction Scheduling

Constraints
- Data dependences
- Resource constraints

Rule
- The final schedule *must* respect these constraints

Dealing with constraints
- How to represent such constraints to deal with during the scheduling process?
Constraints influencing Instruction Scheduling

Constraints
- Data dependences
- Resource constraints

Rule
- The final schedule *must* respect these constraints

Dealing with constraints
- How to represent such constraints to deal with during the scheduling process?
- Data dependences → graph
- Resource constraints → reservation tables or automaton
Data Dependence Representation

Data Dependence Graph (DDG)

- 1 node $\Leftrightarrow$ 1 instruction
- 1 edge $\Leftrightarrow$ 1 flow dependence (directed graph)
- Edge label = parameters of the dependence
  - Latency (# of cycles)
  - Distance (# of iterations)
Data Dependence Representation

Data Dependence Graph (DDG)

- 1 node ⇔ 1 instruction
- 1 edge ⇔ 1 flow dependence (directed graph)
- Edge label = parameters of the dependence
  - Latency (# of cycles)
  - Distance (# of iterations)

Example (1-cycle latency):

\[
\begin{align*}
a &= b + c ; & // \text{ADD1} \\
d &= a + 3 ; & // \text{ADD2} \\
e &= a + d ; & // \text{ADD3}
\end{align*}
\]
Data Dependence Representation

Data Dependence Graph (DDG)

- 1 node ⇔ 1 instruction
- 1 edge ⇔ 1 flow dependence (directed graph)
- Edge label = parameters of the dependence
  - Latency (# of cycles)
  - Distance (# of iterations)

Example (1-cycle latency):

```
  a = b + c ; // ADD1
  d = a + 3 ; // ADD2
  e = a + d ; // ADD3
```
Data Dependence Representation – Example 2

- **Daxpy loop:** *double alpha times X plus Y*
  - \( y \leftarrow \alpha \times x + y \)

- **C-like code:**
  ```c
  for ( i=0; i<N; i++)
      Y[i] = alpha*X[i] + Y[i];
  ```

- **Targeting Itanium ISA:**
  - **LD:** Load from memory (latency 6 cycles from L2 cache)
  - **ST:** Store to memory
  - **FMA:** Fuse multiply and add (latency 4 cycles)
Data Dependence Representation – Example 2

- Daxpy loop: *double alpha times X plus Y*
  - \( y \leftarrow \alpha \times x + y \)
- C-like code:
  ```c
  for ( i=0; i<N; i++)
    Y[i] = alpha*X[i] + Y[i];
  ```
- Targeting Itanium ISA:
  - **LD**: Load from memory (latency 6 cycles from L2 cache)
  - **ST**: Store to memory
  - **FMA**: Fuse multiply and add (latency 4 cycles)
Data Dependence Representation – Example 3

- Daxpy loop with inter-iteration dependence
- C-like code:
  ```
  for ( i=0; i<N; i++)
      Y[i+2] = alpha*X[i] + Y[i];
  ```
- Inter-iteration dependence
- Distance of 2
Daxpy loop with inter-iteration dependence

C-like code:

```c
for ( i=0; i<N; i++)
    Y[i+2] = alpha*X[i] + Y[i];
```

- Inter-iteration dependence
- Distance of 2
Data Dependence Representation

Remarks
- Circuits allowed for a distance > 0
- For basic block, this is only a DAG

Drawbacks
- One fix digit for latency
  - Fixed latencies
  - May not be suitable for cache/memory accesses
- One digit for the distance
  - Only uniform dependences
# Resource Constraint Representation

<table>
<thead>
<tr>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second set of constraints: resource usage/assignment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need to check if two instructions may race for the same resource (functional unit, bus, pipeline stage, . . . )</td>
</tr>
<tr>
<td>Can be several cycles ahead (latency &gt; 1)</td>
</tr>
</tbody>
</table>
Resource Constraint Representation

Resources
- Second set of constraints: resource usage/assignment

Overview
- Need to check if two instructions may race for the same resource (functional unit, bus, pipeline stage, ...)
- Can be several cycles ahead (latency > 1)

State-of-the-art
- 2 representations: reservation tables and automaton
Reservation Tables – Definition

**Reservation tables**
- Intuitive way: resource usage of one instruction as a 2D table

**Semantics**
- Rows: latency of the instruction (in cycles)
- Columns: number of resources available in the target architecture
- Cell \((i,j)\) is marked \(\iff\) instruction requires \(i^{th}\) resource during its \(j^{th}\) cycle of execution
  - Binary tables
- Several tables per instruction (alternatives/options)
Reservation Tables – Example 1

Example with pipelined resources:

- 2 fully pipelined resources (ALU): ALU0 and ALU1
- 2 instructions ADD and MUL
- Constraints:
  - ADD can be executed on ALU0 or ALU1
  - MUL can only be executed on ALU1
Reservation Tables – Example 1

Example with pipelined resources:

- 2 fully pipelined resources (ALU): ALU0 and ALU1
- 2 instructions ADD and MUL
- Constraints:
  - ADD can be executed on ALU0 or ALU1
  - MUL can only be executed on ALU1

<table>
<thead>
<tr>
<th>Tables for ADD:</th>
<th>Table for MUL:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALU0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>OR</td>
<td>ALU0</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Reservation Tables – Example 1

**ADD instruction:**

<table>
<thead>
<tr>
<th></th>
<th>ALU0</th>
<th>ALU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**OR**

<table>
<thead>
<tr>
<th></th>
<th>ALU0</th>
<th>ALU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**MUL instruction:**

<table>
<thead>
<tr>
<th></th>
<th>ALU0</th>
<th>ALU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Are the following sequences valid?

ADD | ADD  
ADD | MUL  
MUL | MUL  
ADD ; ADD 
ADD | MUL ; MUL

ADD | MUL
**Reservation Tables – Example 1**

**ADD instruction:**

<table>
<thead>
<tr>
<th></th>
<th>ALU0</th>
<th>ALU1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**OR**

<table>
<thead>
<tr>
<th></th>
<th>ALU0</th>
<th>ALU1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**MUL instruction:**

<table>
<thead>
<tr>
<th></th>
<th>ALU0</th>
<th>ALU1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

- Are the following sequences valid?
  - ADD | ADD
  - ADD | MUL
  - MUL | MUL
  - ADD ; ADD
  - ADD | MUL ; MUL
### Reservation Tables – Example 1

**ADD instruction:**

<table>
<thead>
<tr>
<th></th>
<th>ALU0</th>
<th>ALU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**OR**

<table>
<thead>
<tr>
<th></th>
<th>ALU0</th>
<th>ALU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**MUL instruction:**

<table>
<thead>
<tr>
<th></th>
<th>ALU0</th>
<th>ALU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

- Are the following sequences valid?
  - ADD | ADD
    - √
  - ADD | MUL
    - √
  - MUL | MUL
    - ×
  - ADD ; ADD
    - √
  - ADD | MUL ; MUL
    - √

- Test if instructions can be scheduled together: AND operation
- Update resource usage: OR operation
Reservation Tables – Example 2

Example with complex resources:

- 2 resources: ALU and LD/ST
- 3 instructions ADD, SUB and LD
- Constraints:
  - ADD instructions have a latency of 1 cycle
  - SUB instructions have a latency of 2 cycles
  - LD uses first the ALU for 1 cycle and then the LD/ST resource for 1 cycle
Reservation Tables – Example 2

Example with complex resources:

- 2 resources: ALU and LD/ST
- 3 instructions ADD, SUB and LD
- Constraints:
  - ADD instructions have a latency of 1 cycle
  - SUB instructions have a latency of 2 cycles
  - LD uses first the ALU for 1 cycle and then the LD/ST resource for 1 cycle

<table>
<thead>
<tr>
<th>Table for ADD:</th>
<th>Table for SUB:</th>
<th>Table for LD:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ALU</td>
<td>LD/ST</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>X</td>
</tr>
</tbody>
</table>
Reservation Tables – Example 2

**ADD instruction:**

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**SUB instruction:**

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**LD instruction:**

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Are the following sequences valid?

- ADD | SUB
- ADD | ADD
- SUB | LD
- LD ; ADD
- LD ; SUB
- SUB ; LD
- ADD ; SUB ; LD
- LD ; ADD ; SUB
### Reservation Tables – Example 2

**ADD instruction:**

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**SUB instruction:**

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**LD instruction:**

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Are the following sequences valid?

- ADD | SUB
- ADD | ADD
- SUB | LD
- LD ; ADD
- LD ; SUB
- SUB ; LD
- ADD ; SUB ; LD
- LD ; ADD ; SUB

- **√** Valid
- **×** Invalid
Reservation Tables – Example 2

ADD instruction:

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

SUB instruction:

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

LD instruction:

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

• Are the following sequences valid?

- ADD | SUB  ×
- ADD | ADD  ×
- SUB | LD   ×
- LD ; ADD  ✓
- LD ; SUB  ✓
- SUB ; LD  ×
- ADD ; SUB ; LD  ×
- LD ; ADD ; SUB  ✓

• Test and update according to latencies of instructions
Reservation Table – Summary

Use
- AND operation to check if several instruction can be scheduled
- OR operation to update the resource state

Advantages
- Intuitive representation
- Small storage

Drawbacks
- Many tests
- Redundant information
Automaton

**Insight**

- Pre-processing of possible resource usages

**Semantics**

- 1 state of the automaton ⇔ 1 assignment of resources
- 1 transition of the automaton ⇔ scheduling of an instruction at the current cycle

**Transition label**

- Label of a transition: the instruction to schedule
- Special label: NOP instruction to advance the current cycle
Automaton – Example 1

**ADD instruction:**

<table>
<thead>
<tr>
<th>ALU0</th>
<th>ALU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

**OR**

<table>
<thead>
<tr>
<th>ALU0</th>
<th>ALU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

**MUL instruction:**

<table>
<thead>
<tr>
<th>ALU0</th>
<th>ALU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>
Automaton – Example 1

ADD instruction:

<table>
<thead>
<tr>
<th></th>
<th>ALU0</th>
<th>ALU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

OR

MUL instruction:

<table>
<thead>
<tr>
<th></th>
<th>ALU0</th>
<th>ALU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

- 2 fully-pipelined resources ⇒ 2 bits per state
Are the following sequences valid?

- ADD | ADD
- ADD | MUL
- MUL | MUL
- ADD; ADD
- ADD | MUL; MUL
Are the following sequences valid?

- ADD | ADD  ✓
- ADD | MUL  ✓
- MUL | MUL  ✗
- ADD ; ADD  ✓
- ADD | MUL ; MUL  ✓
## Automaton – Example 2

**ADD instruction:**

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**SUB instruction:**

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**LD instruction:**

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Automaton – Example 2

### ADD instruction:

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

### SUB instruction:

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

### LD instruction:

<table>
<thead>
<tr>
<th></th>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Are the following sequences valid?

- ADD | SUB  ?
- ADD | ADD  ?
- SUB | LD   ?
- LD ; ADD  ?
- LD ; SUB  ?
- LD ; ADD ; SUB  ?
- LD ; SUB ; LD  ?
- ADD ; SUB ; LD  ?
Are the following sequences valid?

- ADD | SUB \(\times\)
- ADD | ADD \(\times\)
- SUB | LD \(\times\)
- LD ; ADD \(\checkmark\)
- LD ; SUB \(\checkmark\)
- SUB ; LD \(\times\)
- ADD ; SUB ; LD \(\times\)
Automaton – Summary

Use
- An instruction can be currently scheduled if there is an output arc from the current state labeled with this instruction
- Update the state by following this arc

Advantages
- Low query time: table lookup

Drawbacks
- Huge computational time (offline)
- Large storage
  - split into several automata
- Not very flexible
  - e.g. hard to schedule instructions not cycle-wise
Scheduling Process

Scheme of a classical scheduler

- High-level part: main heuristic taken care of the data dependences and driving the scheduling process
- Low-level part: storage of the resource usages and updates of the global assignments
Scheduling Process

Scheme of a classical scheduler

- High-level part: main heuristic taken care of the data dependences and driving the scheduling process
- Low-level part: storage of the resource usages and updates of the global assignments

Scheduling process

- Process begins in the high-level part
- Pick up the next instruction to insert in the partial schedule
- Query the low-level part for resource assignments:
  - If okay, then goes on with another instruction
  - Otherwise backtrack
Acyclic Scheduling: List Scheduling

Context
- Schedule a basic block \(\Rightarrow\) acyclic scheduling
- Goal: minimize the length of the generated code
- Must respect data dependences and resource constraints

Example
- Sum the first element of 3 vectors \(X, Y\) and \(Z\) in the first cell of array \(A\):
  \[
  A[0] = X[0] + Y[0] + Z[0];
  \]
- 3 instructions: ADD, LD, ST (1-cycle latency)
- 3 fully-pipelined resources: ALU, LD0 and LD/ST1 units
Acyclic Scheduling – Example

DDG?
Acyclic Scheduling – Example

DDG:

Reservation tables:
Acyclic Scheduling – Example

**DDG:**

**Reservation tables:**

**ADD instruction:**

<table>
<thead>
<tr>
<th>ALU</th>
<th>LD0</th>
<th>LD/ST1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**LD instruction:**

<table>
<thead>
<tr>
<th>ALU</th>
<th>LD0</th>
<th>LD/ST1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**ST instruction:**

<table>
<thead>
<tr>
<th>ALU</th>
<th>LD0</th>
<th>LD/ST1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Acyclic Scheduling – Example

DDG:

Reservation tables:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>ALU</th>
<th>LD0</th>
<th>LD/ST1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD instruction:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD instruction:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST instruction:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

A possible schedule?
Acyclic Scheduling – Example

A possible schedule respecting both constraints and minimizing the total length:

LD(X) | LD(Y) ;  // Cycle 1
ADD1 | LD(Z) ;  // Cycle 2
ADD2 ;  // Cycle 3
ST ;  // Cycle 4 = length
A possible schedule respecting both constraints and minimizing the total length:

```
LD(X) | LD(Y) ; // Cycle 1
ADD1 | LD(Z) ;  // Cycle 2
ADD2 ;      // Cycle 3
ST ;       // Cycle 4 = length
```

- Good the execute as much instructions as possible
- Pick up the good instruction is crucial (LD(X) and LD(Y) before LD(Z))
- Be careful of explicit resource assignments through reservation tables:
  - Only one valid combination to execute a ST and a LD at the same cycle
List Scheduling

**Principle**
- List scheduling algorithm is based on this approach
- Sort the instruction according to priority based on data dependences
- Pick up one ready instruction in priority order
- Until every instruction has been scheduled

**Priority**
- Many priority schemes exist
- We will use the *height-based priority*:
  - Priority of a node is the longest path from that node to the furthest leaf
  - The path is weighted by latencies
Conclusion

Instruction scheduling
- Generate a total order of a set of instructions

Constraints
- Data dependences
  - Represented as a graph: DDG
- Resource usages
  - Represented as reservation tables or automaton

Acyclic scheduling
- List scheduling
- Assign priority to instructions according to their contribution to the critical path