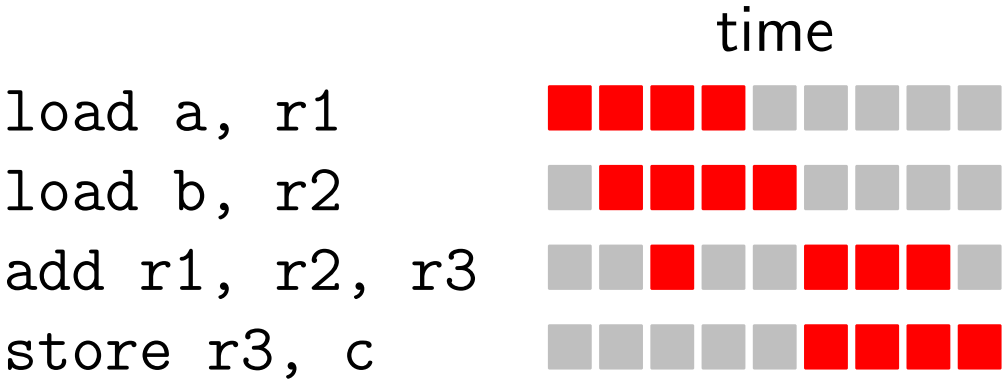
Pipelining



Hazards

Data hazard: instruction uses result of earlier instruction

```
r1 = load r4
r3 = r1 + r2
```

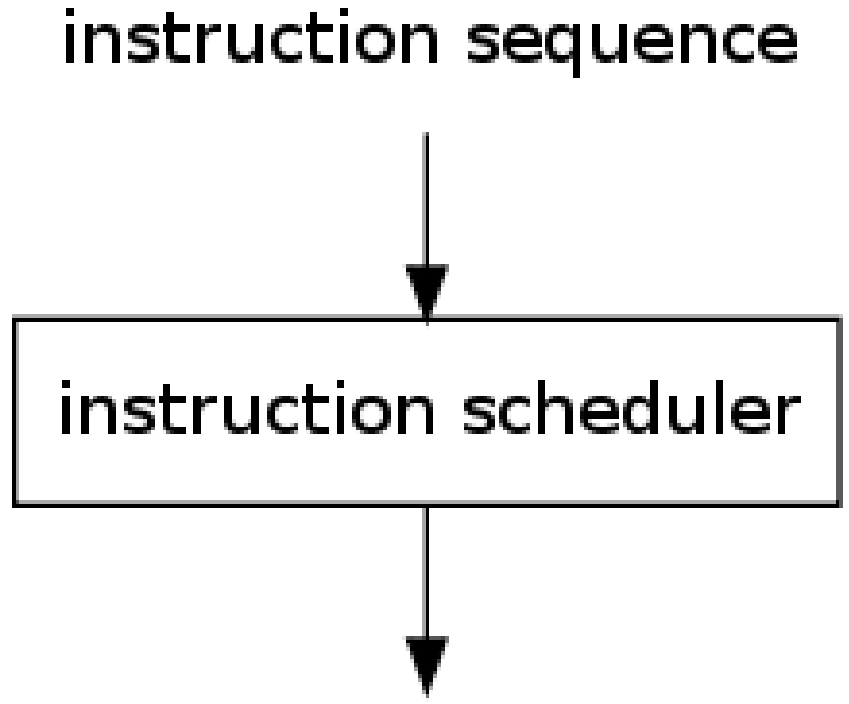
Control hazard: instruction execution conditional on result of earlier instruction

```
if r1 goto label
r4 = r2 + r3
```

Structural hazard: instruction requires resources in use by earlier instruction

```
r1 = r2 * r3
r4 = r5 * r6
```

Instruction Scheduling



same set of instructions
different order
same semantics

Dependences

An instruction j occurring after i is dependent on i if moving j before i would change the semantics of the program.

- Control dependence: i determines whether j executes
- Data dependences:
  - True dependence (WR):
    i computes value, j uses it

    i: x = ...
    j: ... = x
  - Output dependence (WW):
    i writes value, j overwrites it

    i: x = ...
    j: x = ...
  - Anti-dependence (RW):
    i reads value, j overwrites it

    i: ... = x
    j: x = ...
  - Input dependence (RR):
    i reads value, j reads it too

    i: ... = x
    j: ... = x

Dependence Graph

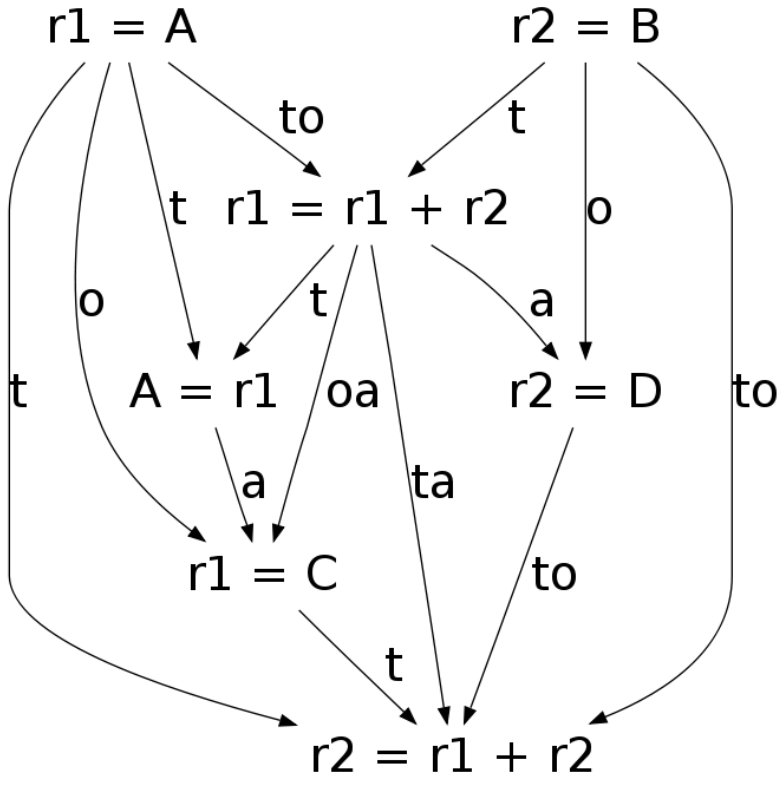
- each instruction is a node
- edge $i \rightarrow j$ if j depends on i

```
r1 = A
r2 = B
r1 = r1 + r2
A = r1
r1 = C
r2 = D
r2 = r1 + r2
```

Dependence Graph

- each instruction is a node
- edge $i \rightarrow j$ if j depends on i

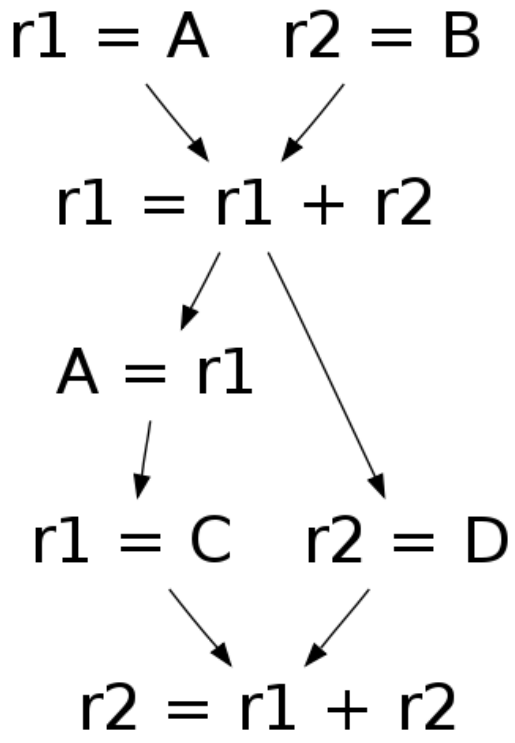
```
r1 = A
r2 = B
r1 = r1 + r2
A = r1
r1 = C
r2 = D
r2 = r1 + r2
```



Dependence Graph

- each instruction is a node
- edge $i \rightarrow j$ if j depends on i

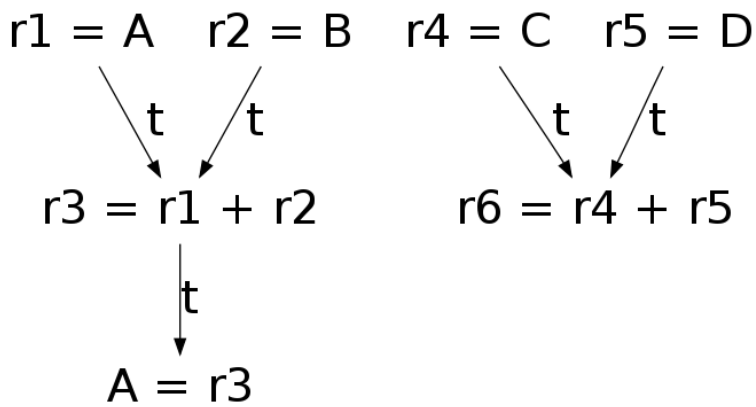
```
r1 = A
r2 = B
r1 = r1 + r2
A = r1
r1 = C
r2 = D
r2 = r1 + r2
```



Dependence Graph

- each instruction is a node
- edge $i \rightarrow j$ if j depends on i

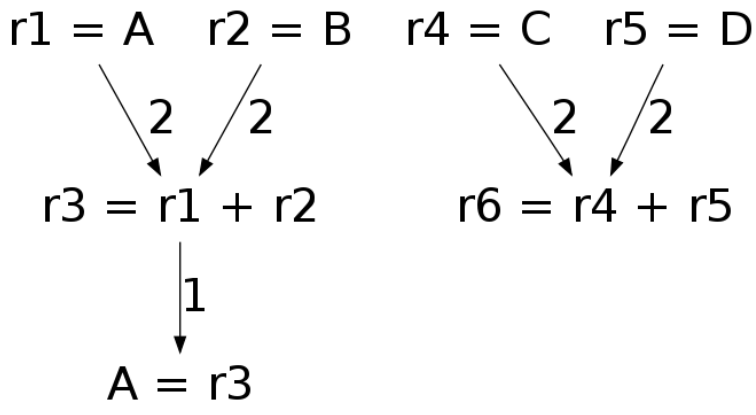
```
r1 = A
r2 = B
r3 = r1 + r2
A = r3
r4 = C
r5 = D
r6 = r4 + r5
```



Dependence Graph

- each instruction is a node
- edge $i \rightarrow j$ if j depends on i

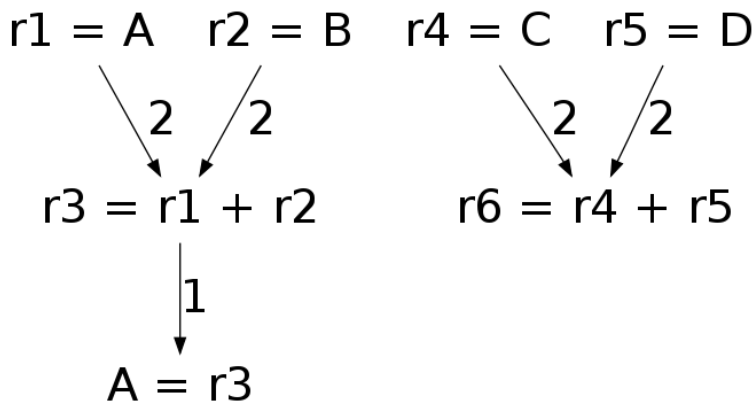
```
r1 = A
r2 = B
stall
r3 = r1 + r2
A = r3
r4 = C
r5 = D
stall
r6 = r4 + r5
```



Dependence Graph

- each instruction is a node
- edge $i \rightarrow j$ if j depends on i

```
r1 = A
r2 = B
r4 = C
r5 = D
r3 = r1 + r2
A = r3
r6 = r4 + r5
```



List Scheduling

Algorithm SCHEDULE():

1: assign each instruction a priority (heuristic)
2: create a list (priority queue) of eligible instructions
   - all predecessors already scheduled
   - all delays already satisfied
3: **repeat**
4: remove highest-priority instruction from list
5: add it to schedule
6: add newly-eligible instructions to list
7: **until** all instructions have been scheduled

Priority Heuristics

- longest (latency-weighted) path from n to a sink node
- number of immediate successors
- number of descendants (not necessarily immediate)
- latency of n
- increase priority for last use of a value
- ...

Instruction Scheduling vs. Register Allocation

- Register allocation may introduce dependences

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

```
r1 = r2 + r3
r2 = r4 + r5
```

Also, any spills need to be scheduled.

- Instruction scheduling may increase register pressure

```
a = ...
... = a
```

```
a = ...
other instructions
... = a
```

Typical solution: schedule, allocate registers, schedule again

Software Pipelining

```
for(i=0; i<100; i++) {
  r1 = a[i];
  r2 = b[i];
  r3 = r1 + r2;
  c[i] = r3;
}
```

Software Pipelining

```
for(i=0; i<100; i++) {
  r1 = a[i];
  r2 = b[i];
  r3 = r1 + r2;
  c[i] = r3;
}
```

```
r1 = a[0];
r2 = b[0];
for(i=0; i<99; i++) {
  r4 = a[i+1];
  r5 = b[i+1];
  r3 = r1 + r2;
  c[i] = r3;
  r1 = r4;
  r2 = r5;
}
r3 = r1 + r2;
c[99] = r3;
```

Scheduling for VLIW Processors

Example CPU: 2 integer, 2 floating point, and 1 branch instruction per cycle.

In each time slot, schedule the best 2 integer, the best 2 fp, and the best branch instruction from the list.

Hardware Scheduling

Many processors support out-of-order execution.

Advantages of OOE

- dependencies and latency may not be statically known
- code may run on different hardware
- hardware register renaming

Advantages of Static Instruction Scheduling

- OOE limited to short dependence paths
- compiler has more time to find good schedule
- hardware resource constraints

For best performance, combine both.