Compiler Construction

Instruction Scheduling

Preserving and computing dependencies?

We construct a directed acyclic graph (DAG) to represent the dependencies between instructions:

- For each instruction in the basic block, create a corresponding vertex in the graph
- For each dependency between two instructions, create a corresponding (annotated) edge in the graph. Note that this edge is annotated.

Computing the dependency graph

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```

Computing the dependency graph

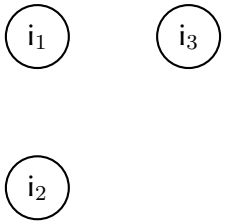
```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```

Computing the dependency graph

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

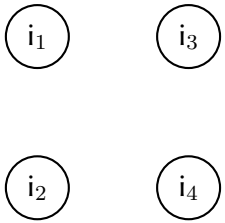
```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



Computing the dependency graph

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

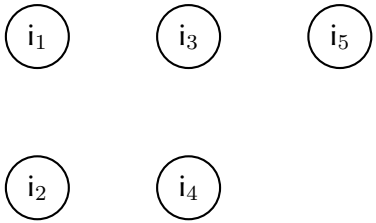
```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



Computing the dependency graph

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

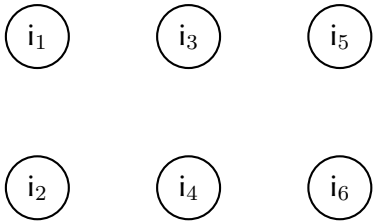
```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



Computing the dependency graph

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

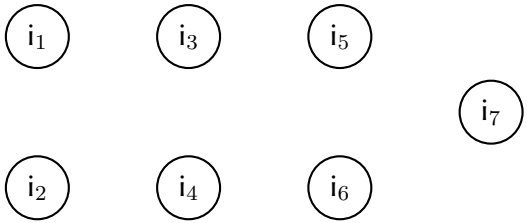
```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



Computing the dependency graph

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

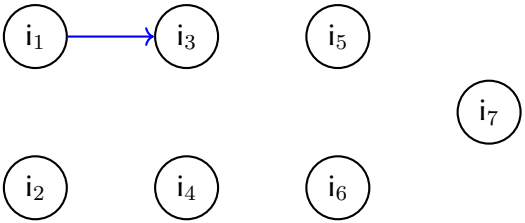
```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



Computing the dependency graph

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```

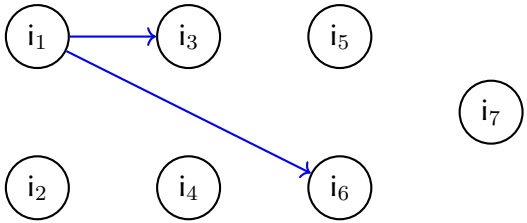


Type of dependency: RAW, WAW, WAR

Computing the dependency graph

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```

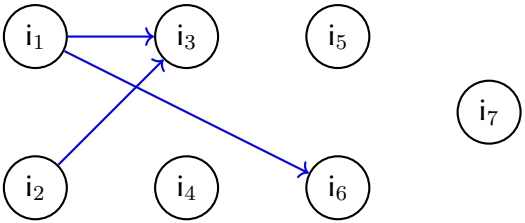


Type of dependency: RAW, WAW, WAR

Computing the dependency graph

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```

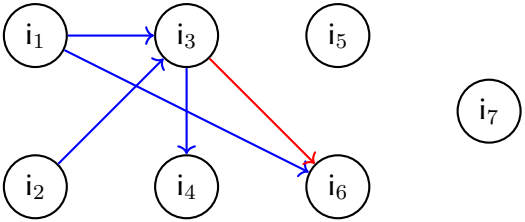


Type of dependency: RAW, WAW, WAR

Computing the dependency graph

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```

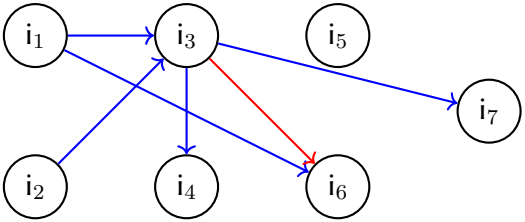


Type of dependency: RAW, WAW, WAR

Computing the dependency graph

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```

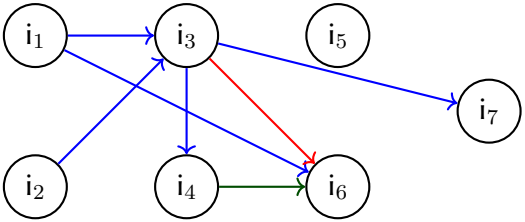


Type of dependency: RAW, WAW, WAR

Computing the dependency graph

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```

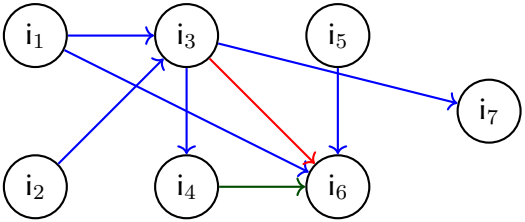


Type of dependency: RAW, WAW, WAR

Computing the dependency graph

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```

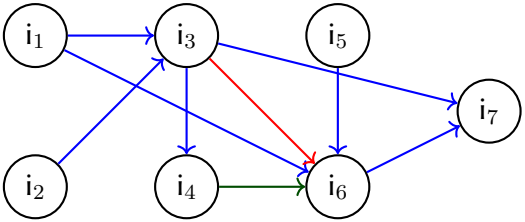


Type of dependency: RAW, WAW, WAR

Computing the dependency graph

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```

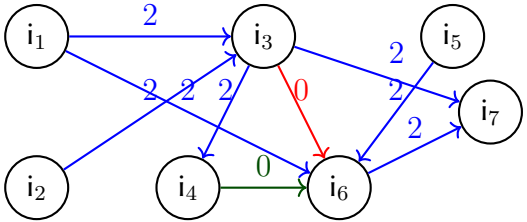


Type of dependency: RAW, WAW, WAR

Preserving dependencies: Critical Path 1/3

The critical path represents the longest path between two nodes. We add **delays** (weights) to edges:

- 0 for WAW and WAR dependencies
- 2 for RAW dependencies with memory access
- 1 for other RAW dependencies



Preserving dependencies: Critical Path 2/3

Any (reverse) topological sort of this DAG (i.e. any linear ordering of the vertices which keeps all the edges "pointing forwards") will maintain the dependencies and hence preserve the correctness of the program.

Preserving dependencies: Critical Path 3/3

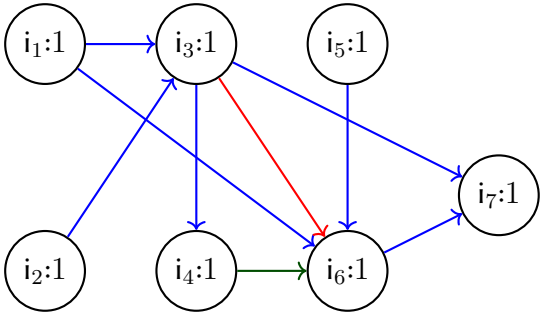
Algorithm:

- Associate a weight 1 to each "instruction node"
- For all nodes n_i in topological postorder
  - If n_i is not a leaf
    - ⋆ For all nodes n_j in succ(n_i) do
      n_i.weight $\leftarrow$ max (n_i.weight, n_j.weight+ delay(n_i, n_j))

Remember "important" edges during computations, they will form the critical path.

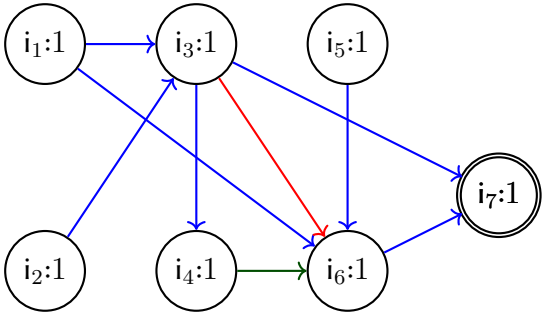
Computing the critical path

Delays: blue arrows 2, red and green 0



Computing the critical path

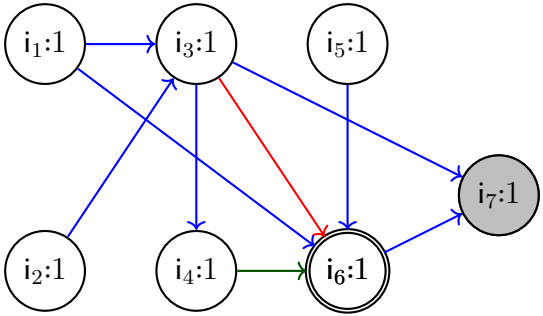
Delays: blue arrows 2, red and green 0



i_7 doesn't have successors, skip it!

Computing the critical path

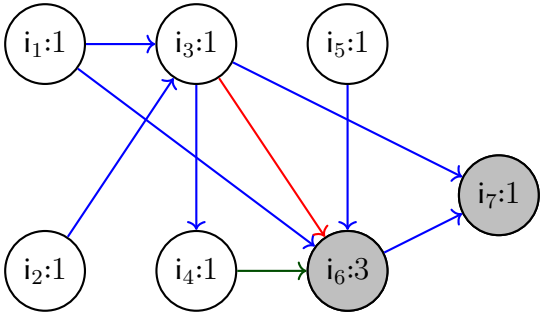
Delays: blue arrows 2, red and green 0



delay(i_6, i_7)=2 $>$ 1, change i_6 weight!

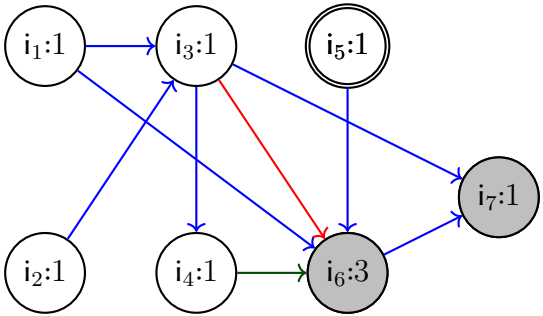
Computing the critical path

Delays: blue arrows 2, red and green 0



Computing the critical path

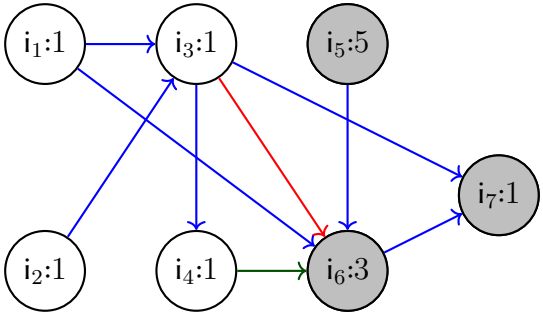
Delays: blue arrows 2, red and green 0



delay(i_5, i_6)=2 $>$ 1, change i_5 weight!

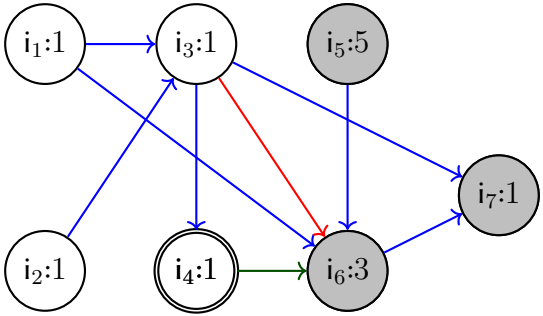
Computing the critical path

Delays: blue arrows 2, red and green 0



Computing the critical path

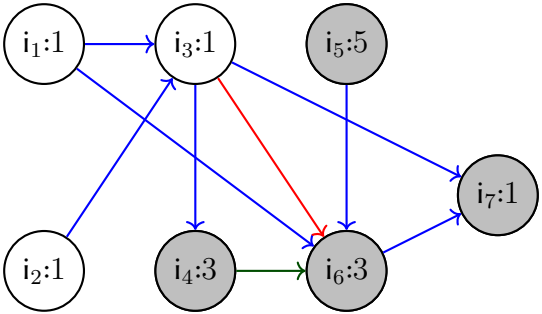
Delays: blue arrows 2, red and green 0



i_6.weight=3 $>$ 1, change i_4 weight!

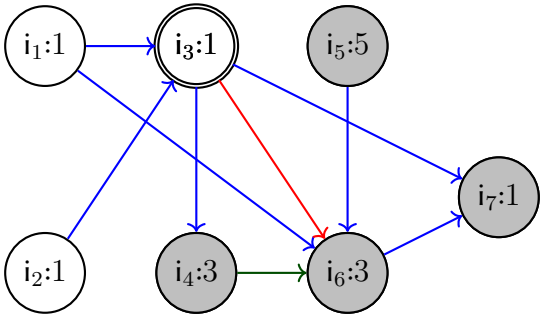
Computing the critical path

Delays: blue arrows 2, red and green 0



Computing the critical path

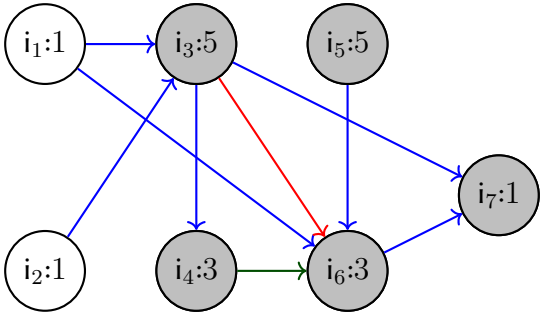
Delays: blue arrows 2, red and green 0



delay(i_3, i_4) + i_4.weight=3 $>$ 1, change i_3 weight!

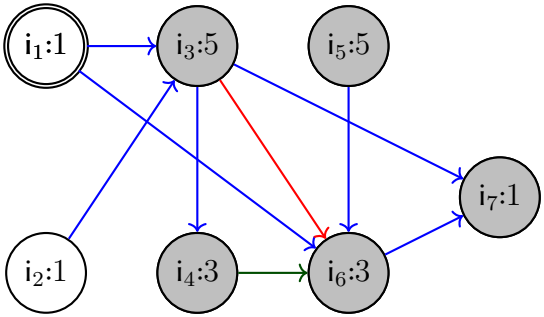
Computing the critical path

Delays: blue arrows 2, red and green 0



Computing the critical path

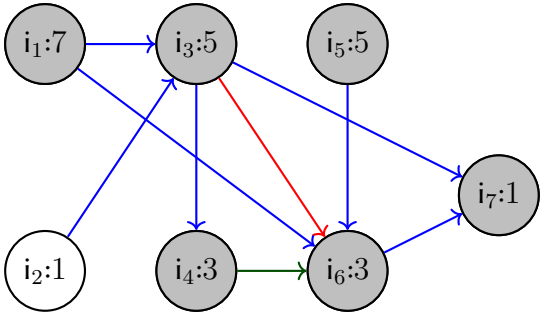
Delays: blue arrows 2, red and green 0



delay(i_1, i_3) + i_3.weight=7 $>$ 1, change i_1 weight!

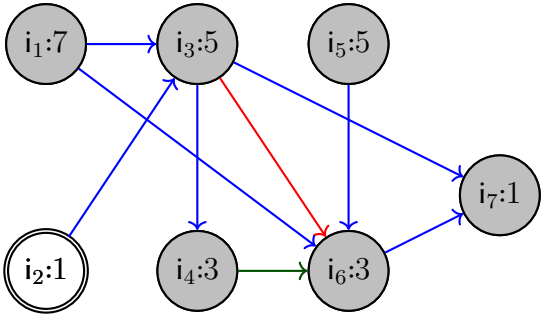
Computing the critical path

Delays: blue arrows 2, red and green 0



Computing the critical path

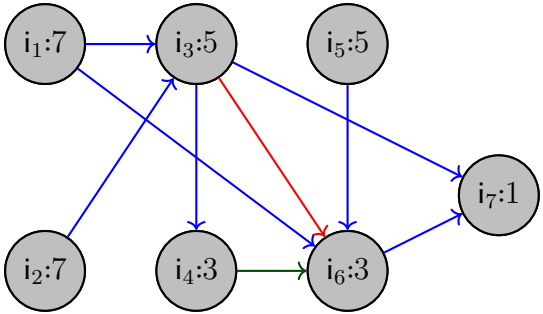
Delays: blue arrows 2, red and green 0



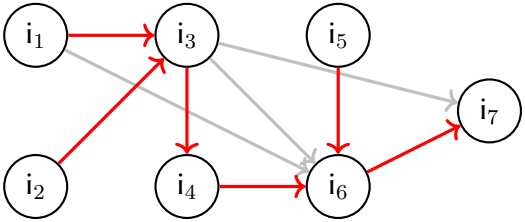
delay(i_2, i_3) + i_3.weight=7 $>$ 1, change i_2 weight!

Computing the critical path

Delays: blue arrows 2, red and green 0



So many orders ...with one critial path



$i_1,i_2,i_3,i_4,i_5,i_6,i_7$	$i_1,i_2,i_3,i_5,i_4,i_6,i_7$
$i_2,i_1,i_3,i_5,i_4,i_6,i_7$	$i_2,i_1,i_3,i_4,i_5,i_6,i_7$
$i_1,i_2,i_5,i_3,i_4,i_6,i_7$	$i_2,i_1,i_5,i_3,i_4,i_6,i_7$
$i_1,i_5,i_2,i_3,i_4,i_6,i_7$	$i_2,i_5,i_1,i_3,i_4,i_6,i_7$
$i_5,i_1,i_2,i_3,i_4,i_6,i_7$	$i_5,i_2,i_1,i_3,i_4,i_6,i_7$

All these permutations respect dependencies
but is there a best instruction scheduling?

Performances and Pipeline

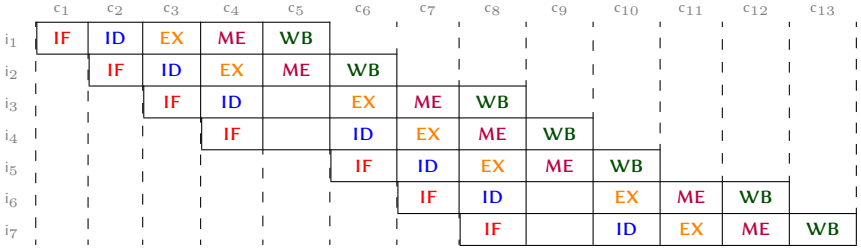
Not all orders are equivalent!

- Some dependencies can bring hazards that slow down performances inside of the pipeline
- Hazard occurs when:
  - 1 instruction requires the previous instruction has finished
  - 2 instructions need the same data at the same time: one of the two is blocked

Back to the example – without scheduling

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



Without scheduling: 2 dependencies, 2 stalls, 13 cycles!

Minimizing Stalls (1/2)

Each time we emit the next instruction, we should try to choose one which

- P_1 does not conflict with the previous emitted instruction
- P_2: is most likely to conflict if first of a pair (e.g. prefer `lw` to `add`)
- P_3: is as far away as possible (along paths in the DAG) from an instruction which can validly be scheduled last

Minimizing Stalls (2/2)

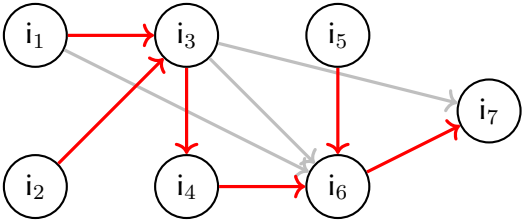
Algorithm:

- Compute the dependency graph
- While the list of candidate instructions is not empty
  - If one instruction satisfies P_1, P_2, and P_3: remove it from the list and emit it.
    - ⋆ Remove the instruction from the DAG and insert the newly minimal elements into the candidate list.
  - Otherwise emit a `nop` instruction

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



Candidates = $\{i_1, i_2, i_5\}$

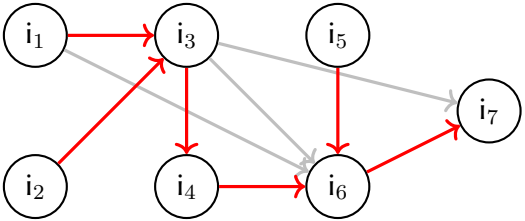
Final Order =

Choose i_1 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



Candidates = $\{i_1, i_2, i_5\}$

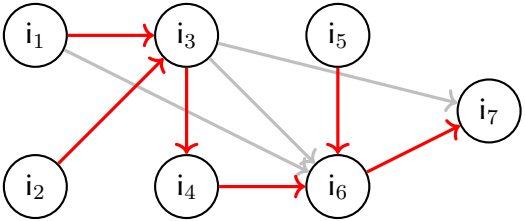
Final Order =

Choose i_1 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



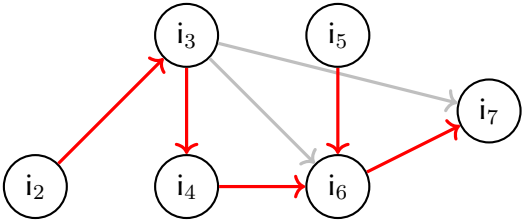
Candidates = $\{i_1, i_2, i_5\}$
Final Order = i_1

Choose i_1 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



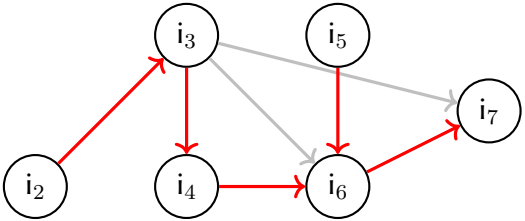
Candidates = $\{i_1, i_2, i_5\}$
Final Order = i_1

Choose i_1 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



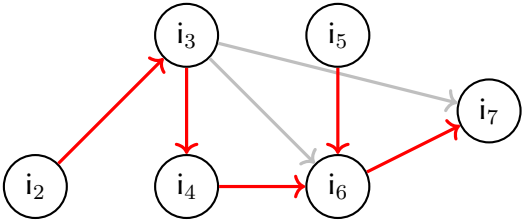
Candidates = $\{i_2, i_5\}$
Final Order = i_1

Choose i_2 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



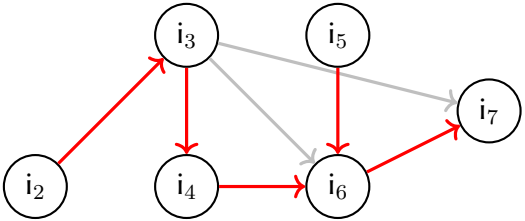
Candidates = $\{i_2, i_5\}$
Final Order = i_1

Choose i_2 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



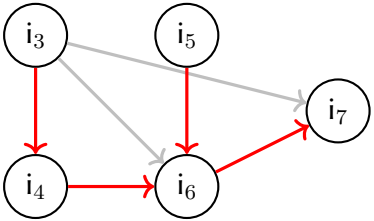
Candidates = $\{i_2, i_5\}$
Final Order = i_1, i_2

Choose i_2 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



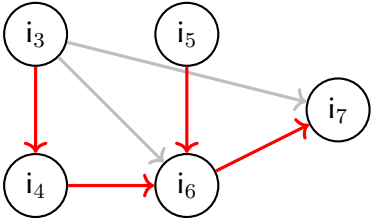
Candidates = $\{i_2, i_5\}$
Final Order = i_1, i_2

Choose i_2 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



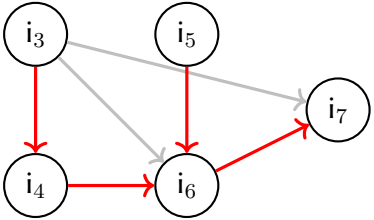
Candidates = $\{i_5, i_3\}$
Final Order = i_1, i_2

Choose i_5 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



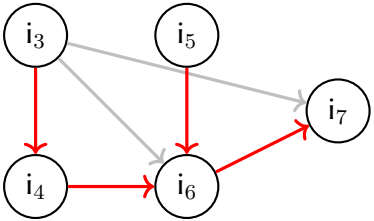
Candidates = $\{i_5, i_3\}$
Final Order = i_1, i_2

Choose i_5 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



Candidates = $\{i_5, i_3\}$

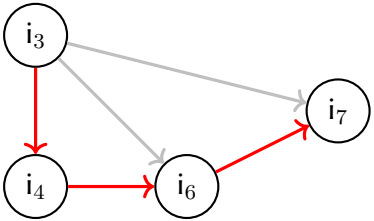
Final Order = i_1, i_2, i_5

Choose i_5 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



Candidates = $\{i_5, i_3\}$

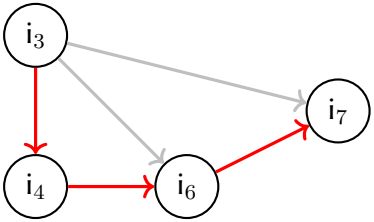
Final Order = i_1, i_2, i_5

Choose i_5 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



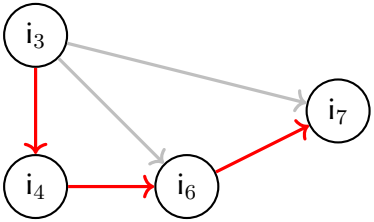
Candidates = $\{i_3\}$
Final Order = i_1, i_2, i_5

Choose i_3 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```



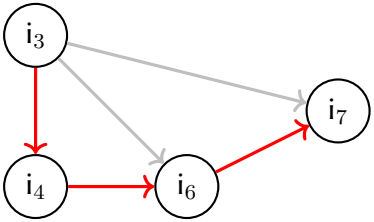
Candidates = $\{i_3\}$
Final Order = i_1, i_2, i_5

Choose i_3 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)
i2 :  lw   $2,4($10)
i3 :  add  $3,$1,$2
i4 :  sw   $3,12($10)
```

```
i5 :  lw   $4,8($10)
i6 :  add  $3,$1,$4
i7 :  sw   $3,16($10)
```

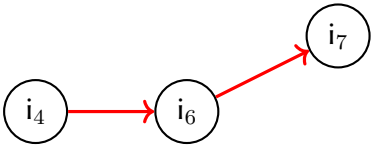


Candidates = $\{i_3\}$
Final Order = i_1, i_2, i_5, i_3

Choose i_3 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

i_1 :	lw	$1,0($10)	i_5 :	lw	$4,8($10)
i_2 :	lw	$2,4($10)	i_6 :	add	$3,$1,$4
i_3 :	add	$3,$1,$2	i_7 :	sw	$3,16($10)
i_4 :	sw	$3,12($10)			



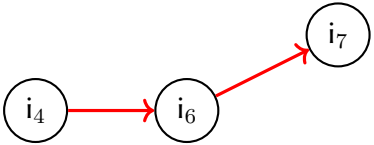
Candidates = $\{i_3\}$

Final Order = i_1, i_2, i_5, i_3

Choose i_3 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)  |  i5 :  lw   $4,8($10)
i2 :  lw   $2,4($10)  |  i6 :  add  $3,$1,$4
i3 :  add  $3,$1,$2   |  i7 :  sw   $3,16($10)
i4 :  sw   $3,12($10) |
```

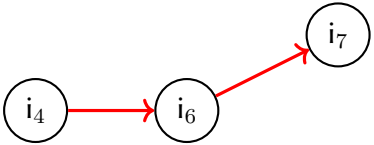


Candidates = $\{i_4\}$
Final Order = i_1, i_2, i_5, i_3

Choose i_4 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)  |  i5 :  lw   $4,8($10)
i2 :  lw   $2,4($10)  |  i6 :  add  $3,$1,$4
i3 :  add  $3,$1,$2   |  i7 :  sw   $3,16($10)
i4 :  sw   $3,12($10) |
```

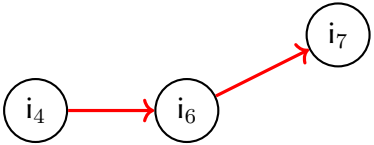


Candidates = $\{i_4\}$
Final Order = i_1, i_2, i_5, i_3

Choose i_4 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)  |  i5 :  lw   $4,8($10)
i2 :  lw   $2,4($10)  |  i6 :  add  $3,$1,$4
i3 :  add  $3,$1,$2   |  i7 :  sw   $3,16($10)
i4 :  sw   $3,12($10) |
```

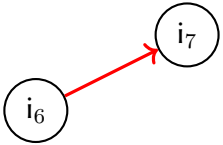


Candidates = $\{i_4\}$
Final Order = i_1, i_2, i_5, i_3, i_4

Choose i_4 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)  |  i5 :  lw   $4,8($10)
i2 :  lw   $2,4($10)  |  i6 :  add  $3,$1,$4
i3 :  add  $3,$1,$2   |  i7 :  sw   $3,16($10)
i4 :  sw   $3,12($10) |
```

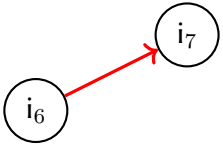


Candidates = $\{i_4\}$
Final Order = i_1, i_2, i_5, i_3, i_4

Choose i_4 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)  |  i5 :  lw   $4,8($10)
i2 :  lw   $2,4($10)  |  i6 :  add  $3,$1,$4
i3 :  add  $3,$1,$2   |  i7 :  sw   $3,16($10)
i4 :  sw   $3,12($10) |
```

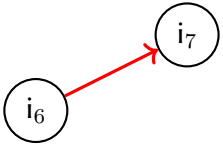


Candidates = $\{i_6\}$
Final Order = i_1, i_2, i_5, i_3, i_4

Choose i_6 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)  |  i5 :  lw   $4,8($10)
i2 :  lw   $2,4($10)  |  i6 :  add  $3,$1,$4
i3 :  add  $3,$1,$2   |  i7 :  sw   $3,16($10)
i4 :  sw   $3,12($10) |
```

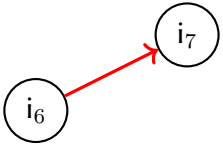


Candidates = $\{i_6\}$
Final Order = i_1, i_2, i_5, i_3, i_4

Choose i_6 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)  |  i5 :  lw   $4,8($10)
i2 :  lw   $2,4($10)  |  i6 :  add  $3,$1,$4
i3 :  add  $3,$1,$2   |  i7 :  sw   $3,16($10)
i4 :  sw   $3,12($10) |
```



Candidates = $\{i_6\}$
Final Order = $i_1, i_2, i_5, i_3, i_4, i_6$

Choose i_6 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)  |  i5 :  lw   $4,8($10)
i2 :  lw   $2,4($10)  |  i6 :  add  $3,$1,$4
i3 :  add  $3,$1,$2   |  i7 :  sw   $3,16($10)
i4 :  sw   $3,12($10) |
```

Candidates = $\{i_6\}$
Final Order = $i_1, i_2, i_5, i_3, i_4, i_6$

Choose i_6 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

i_1 :	lw	$1,0($10)	i_5 :	lw	$4,8($10)
i_2 :	lw	$2,4($10)	i_6 :	add	$3,$1,$4
i_3 :	add	$3,$1,$2	i_7 :	sw	$3,16($10)
i_4 :	sw	$3,12($10)			

Candidates = $\{i_7\}$
Final Order = $i_1, i_2, i_5, i_3, i_4, i_6$

Choose i_7 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

i_1 :	lw	$1,0($10)	i_5 :	lw	$4,8($10)
i_2 :	lw	$2,4($10)	i_6 :	add	$3,$1,$4
i_3 :	add	$3,$1,$2	i_7 :	sw	$3,16($10)
i_4 :	sw	$3,12($10)			

Candidates = $\{i_7\}$
Final Order = $i_1, i_2, i_5, i_3, i_4, i_6$

Choose i_7 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)  |  i5 :  lw   $4,8($10)
i2 :  lw   $2,4($10)  |  i6 :  add  $3,$1,$4
i3 :  add  $3,$1,$2   |  i7 :  sw   $3,16($10)
i4 :  sw   $3,12($10) |
```

Candidates = $\{i_7\}$
Final Order = $i_1, i_2, i_5, i_3, i_4, i_6, i_7$

Choose i_7 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

i_1 :	lw	$1,0($10)	i_5 :	lw	$4,8($10)
i_2 :	lw	$2,4($10)	i_6 :	add	$3,$1,$4
i_3 :	add	$3,$1,$2	i_7 :	sw	$3,16($10)
i_4 :	sw	$3,12($10)			

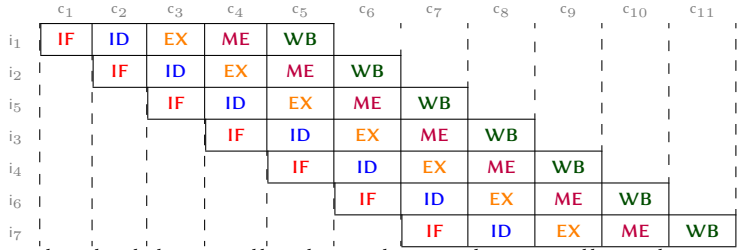
Candidates = $\{i_7\}$
Final Order = $i_1, i_2, i_5, i_3, i_4, i_6, i_7$

Choose i_7 since it satisfies P_1, P_2 and P_3

Applying scheduling algorithm to the example

```
i1 :  lw   $1,0($10)  |  i5 :  lw   $4,8($10)
i2 :  lw   $2,4($10)  |  i6 :  add  $3,$1,$4
i3 :  add  $3,$1,$2   |  i7 :  sw   $3,16($10)
i4 :  sw   $3,12($10) |
```

Final Order = i_1, i_2, i_5, i_3, i_4, i_6, i_7



With scheduling: still 2 dependencies but 0 stalls and 11 cycles!

A word on scheduling strategies

- Sometimes we cannot avoid some stalls
- Computing the critical path can be smarter
- Computing the DAG of dependencies can be done in $O(n^2)$ by scanning backwards through the basic block and adding edges as dependencies arise

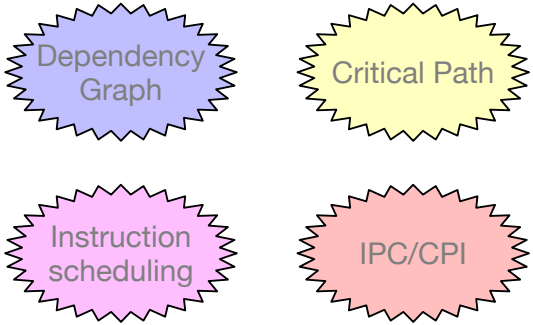
A word on performances

We can statically compute instructions per cycle IPC=$\frac{\text{nb instructions}}{\text{nb cycles}}$, to evaluate 2 possible schedulings.

In the previous example:

- without scheduling IPC=$\frac{7}{13} = 0.53$
- with scheduling IPC=$\frac{7}{11} = 0.63$ (better!)

Summary


Dependency Graph
Critical Path
Instruction scheduling
IPC/CPI