Technische Universiteit
Eindhoven
University of Technology

Backend algorithms

Roel Jordans

Today's program

- Scheduling algorithms
  - List Scheduling
  - Swing Modulo Scheduling
- Register allocation

Instruction level parallelism

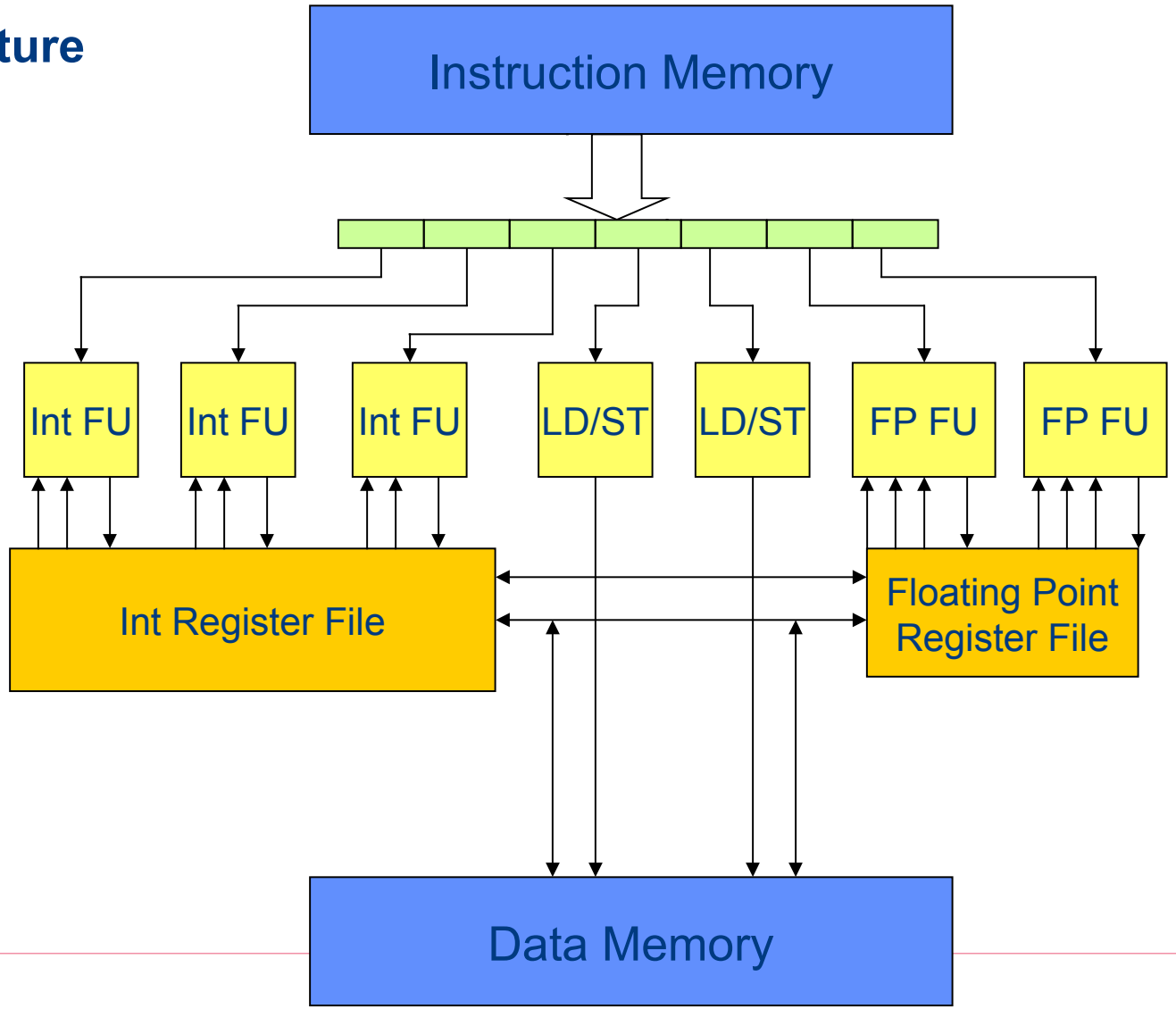
- Execute multiple operations in parallel

  VLIW or Superscalar

- Energy efficient
  - Lower clock frequency → lower working voltage

VLIW general concept

A VLIW architecture with 7 FUs



Instruction scheduling

- Compiler determines order in which the processor gets the instructions
  - Superscalar processors may reorder again
- Should try to utilize the available resources
  - ALU, LSU, MUL?, FPU?

Scheduling: **Overview**

Transforming a sequential program into a parallel program:

```
read sequential program
read machine description file
for each procedure do
    perform function inlining
for each procedure do
    transform an irreducible CFG into a reducible CFG
    perform control flow analysis
    perform loop unrolling
    perform data flow analysis
    perform memory reference disambiguation
    perform register allocation
    for each scheduling scope do
        perform instruction scheduling
write out the parallel program
```

Basic Block Scheduling

Basic Block =
piece of code which can only be entered from the top (first instruction) and left at the bottom (final instruction)

Scheduling a basic block =
Assign resources and a cycle to every operation

List Scheduling =
Heuristic scheduling approach, scheduling the operation one-by-one

Time_complexity = O(N), where N is #operations

Optimal scheduling has Time_complexity = O(exp(N)

Question: what is a good scheduling heuristic

Basic Block Scheduling

Make a **Data Dependence Graph** (DDG)

Determine minimal length of the DDG (for the given architecture)
minimal number of cycles to schedule the graph (assuming sufficient resources)

Determine:
ASAP (As Soon As Possible) cycle = earliest cycle instruction can be scheduled
ALAP (As Late As Possible) cycle = latest cycle instruction can be scheduled
Slack of each operation = ALAP – ASAP

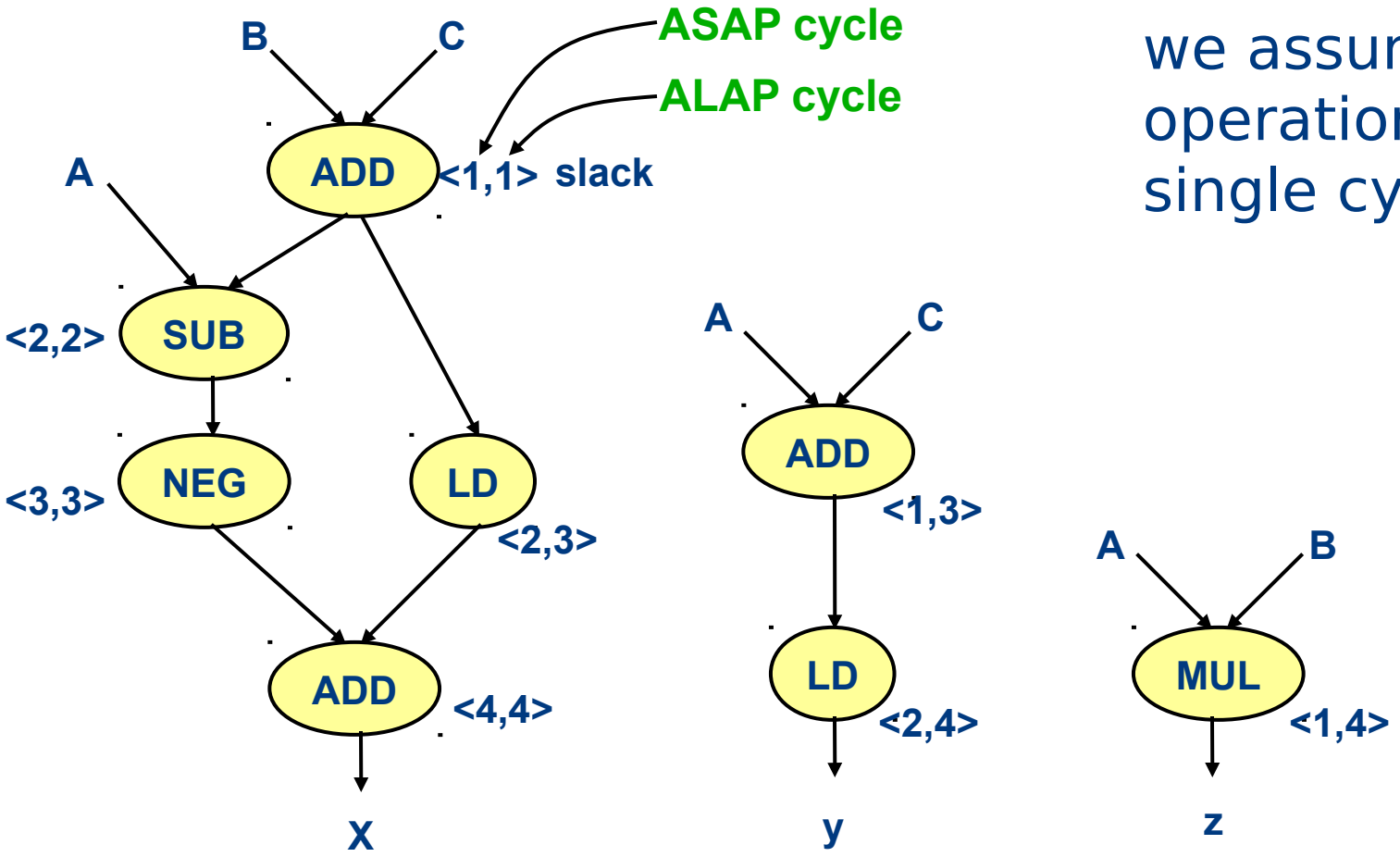
- **Priority of operations = f (Slack, #decendants, #register impact, ….)**

Place each operation in first cycle with sufficient resources

Notes:
Basic Block = a (maximal) piece of consecutive instructions which can only be entered at the first instruction and left at the end

Scheduling order sequential
Scheduling Priority determined by used heuristic; e.g. slack + other contributions

Basic Block Scheduling:
determine ASAP and ALAP cycles



we assume all operations are single cycle !

List scheduler

- Most common scheduler around
- Uses heuristics to select which instruction to schedule next

Cycle based list scheduling

proc Schedule(*DDG* = (*V*,*E*))
beginproc
  ready = { *v* | ¬∃(*u*,*v*) ∈ *E* }
  ready' = *ready*
  sched = ϕ
  current_cycle = 0
  while *sched* ≠ *V* **do**
    for each *v* ∈ *ready' (select in priority order)* **do**
      if ¬ResourceConfl(*v*,*current_cycle*, *sched*) **then**
        cycle(*v*) = *current_cycle*
        sched = *sched* ∪ {*v*}
      endif
    endfor
    current_cycle = *current_cycle* + 1
    ready = { *v* | *v* ∉ *sched* ∧ ∀ (*u*,*v*)∈ *E*, *u* ∈ *sched* }
    ready' = { *v* | *v* ∈ *ready* ∧ ∀ (*u*,*v*)∈ *E*, *cycle(u)* + *delay*(*u*,*v*) ≤ *current_cycle*}
  endwhile
endproc

The algorithm

- Construct a work-list of nodes that are ready
- Pick the node with highest priority for which we have an *execution resource available*
  - *Selection* according to the heuristic
- Schedule that node at current cycle
  - *Top-down or bottom-up*
- Repeat steps until finished

Resource model

- Model of the processor in current cycle
  - Uses Deterministic Finite Automata (DFA)
- Used to test which execution resources are still available
  - As you will find out in assignment 2

Commonly used heuristics

- Many existing options
  - Earliest start time
  - Latest start time
  - # children
  - # registers born
  - # registers killed
- See reading material at the end!

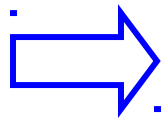
Peephole optimizations

- Fixing small mistakes in the generated code
  - Optimize while only looking at a small part of the generated code
- Correcting for greedy behaviour of scheduler
  - Moving operations with slack to a later time to reduce register pressure
- Correcting inefficient instruction selection
  - Late operation combining
    - Fix things like assigment 1b

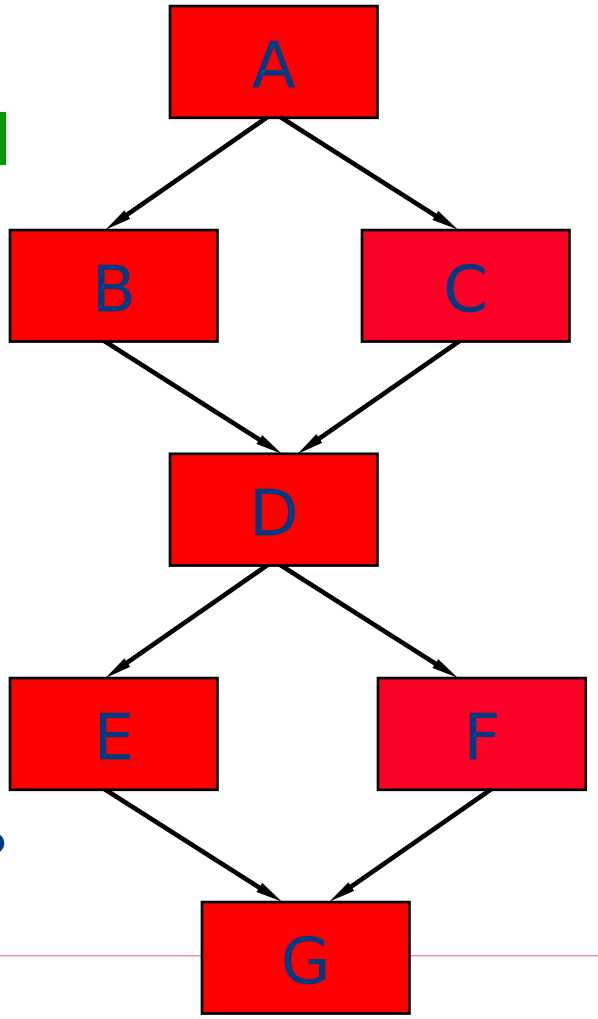
Extended Scheduling Scope: look at the CFG

Code:

```
A;
If cond
  Then B
  Else  C;
D;
If cond
  Then E
  Else  F;
G;
```



CFG: Control Flow Graph



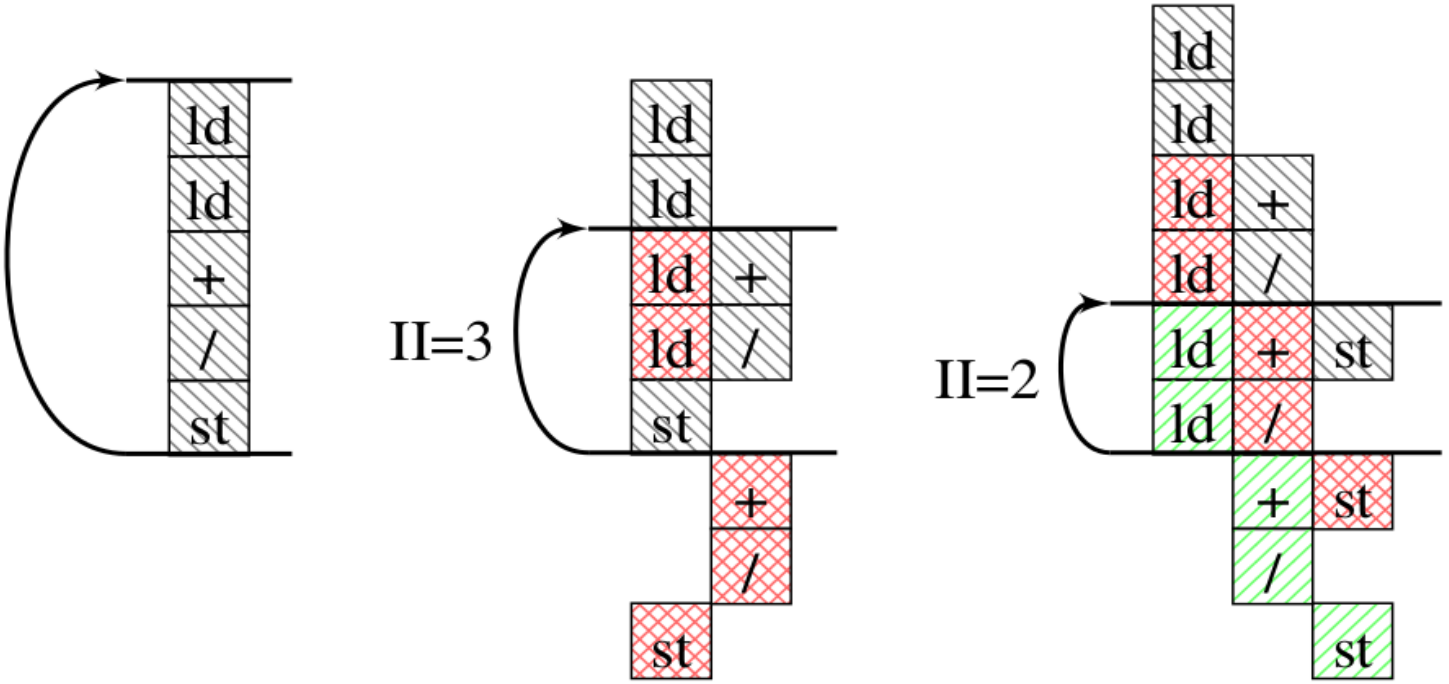
Q: Why enlarge the scheduling scope?

Modulo scheduling

- Interleave operations from multiple *loop* iterations
  - Improved loop ILP
  - Overlapping execution of loops, new iterations start with an *initiation interval* (II)
- Requires both loop dependency and resource availability information

Resource constraints

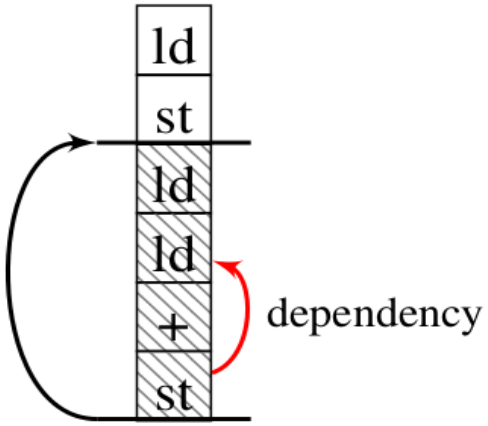
```
for(int i = 0; i < N; i++) {
    B[i] = (A[2*i] + A[2*i+1]) / 2;
}
```



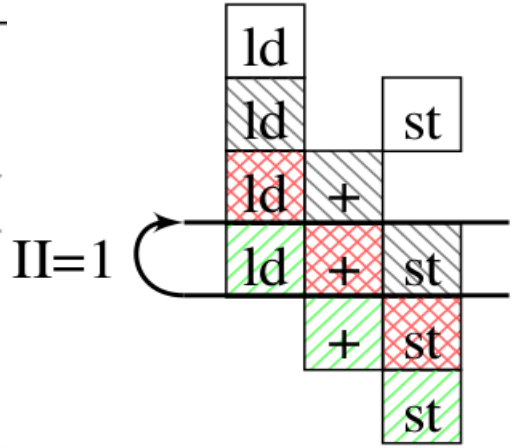
(a) Original (b) Single memory (c) Two memories

Data dependencies

```
B[0] = A[0];
for(int i = 1; i < N; i++) {
    B[i] = B[i-1] + A[i];
}
```




```
register int r = A[0];
B[0] = r;
for(int i = 1; i < N; i++) {
    r = r + A[i];
    B[i] = r;
}
```



Modulo scheduling constraints

MII, minimum initiation interval, bounded by cyclic dependences and resources:

$$MII = \max\{ ResMinII, RecMinII \}$$

Resources:

$$\text{ResMinII} = \max_{r \in resources} \left\lceil \frac{used(r)}{available(r)} \right\rceil$$

Cycles:

$$cycle(v) \geq cycle(v) + \sum_{e \in c} \{ delay(e) - II.distance(e) \}$$

Therefore:

$$\text{RecMinII} = \min \left\{ II \in N \mid \forall_{c \in cycles}, 0 \geq \sum_{e \in c} \{ delay(e) - II.\text{distance}(e) \} \right\}$$

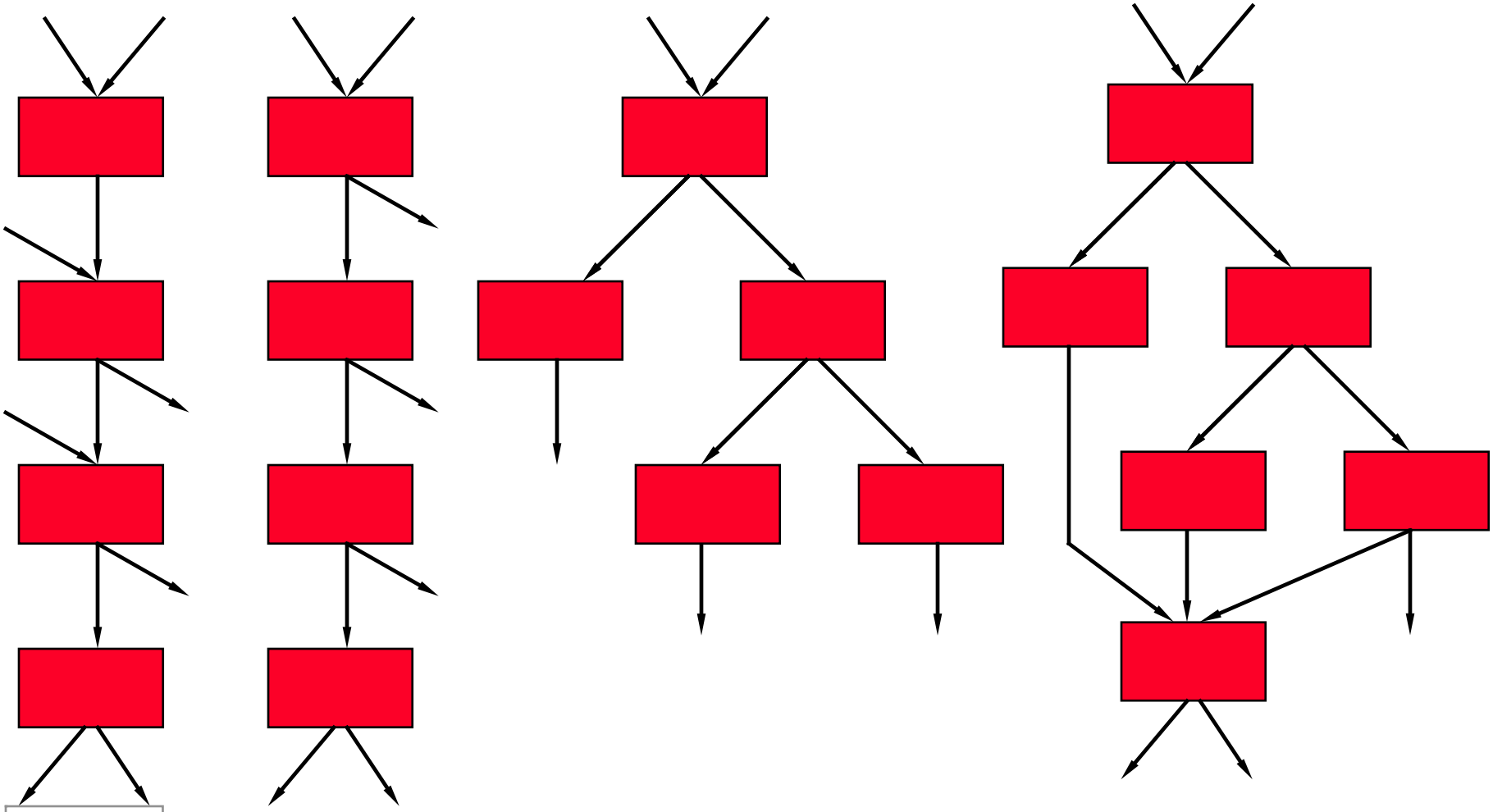
Or:

$$\text{RecMinII} = \max_{c \in cycles} \left\lceil \frac{\sum_{e \in c} delay(e)}{\sum_{e \in c} \text{distance}(e)} \right\rceil$$

Swing-modulo scheduling

- Fast heuristic algorithm
  - Used in many compilers (GCC, LLVM, …)
- Scheduling in 5 steps
  - Find cyclic (loop carried) dependencies and their length
  - Find resource pressure
  - Comupute minimal initiation interval
  - Order nodes according to criticality
  - Schedule nodes in that order
    - If failed, increase II and retry
- More info in the reading material

Other Scheduling Scopes



Register allocation

Given a set of registers, what is the most efficient

mapping of registers to program variables in terms

of execution time of the program?

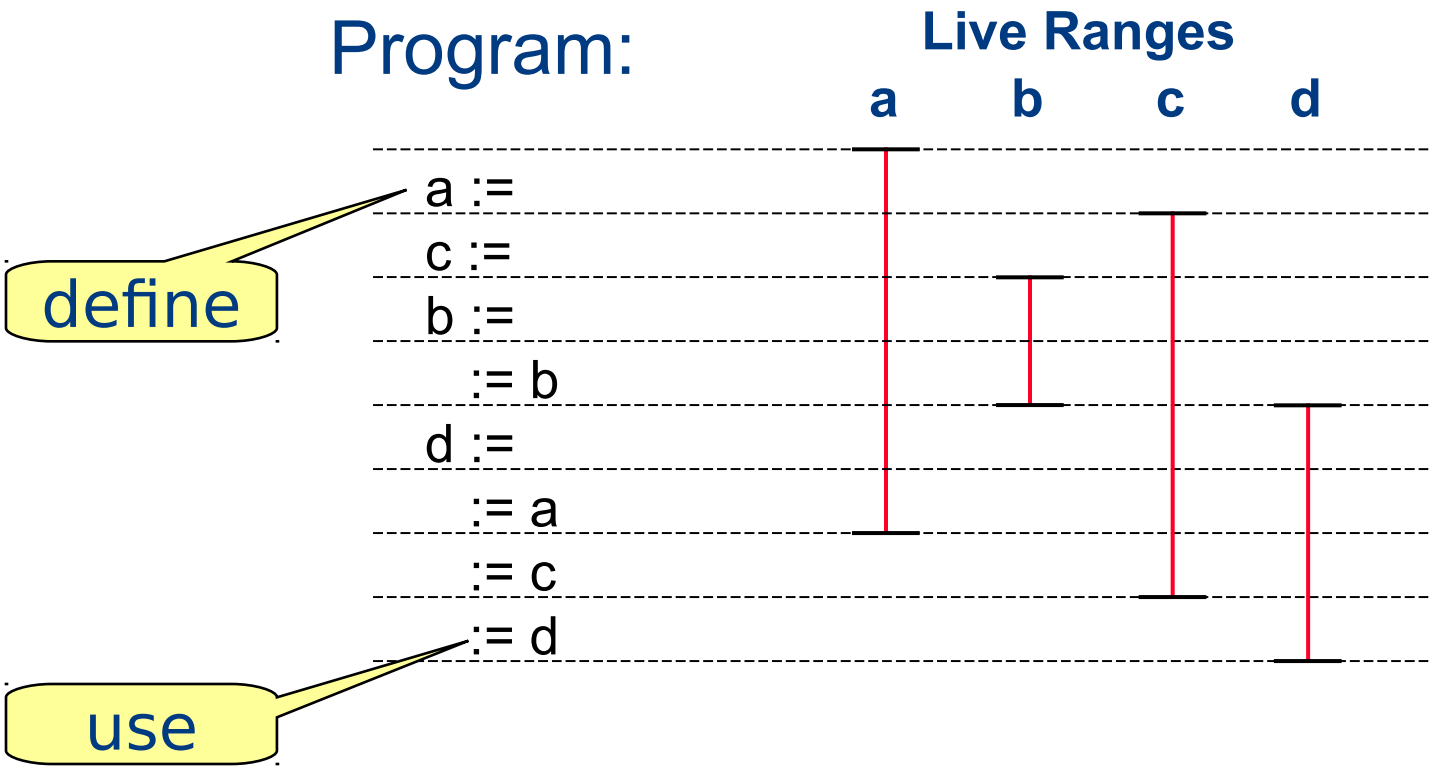
Some definitions:

A variable is ***defined*** at a point in program when a value is assigned to it.

A variable is ***used*** at a point in a program when its value is referenced in an expression.

The ***live range*** of a variable is the execution range between definitions and uses of a variable.

Register allocation using graph coloring

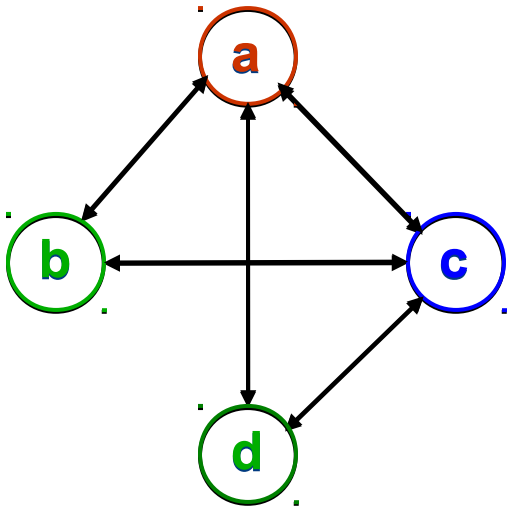


Register allocation using graph coloring

Inference Graph

Coloring:

- a = red
- b = green
- c = blue
- d = green



Graph needs 3 colors => program needs 3 registers

Question: map coloring requires (at most) 4 colors; what's the maximum number of colors (= registers) needed for register interference graph coloring?

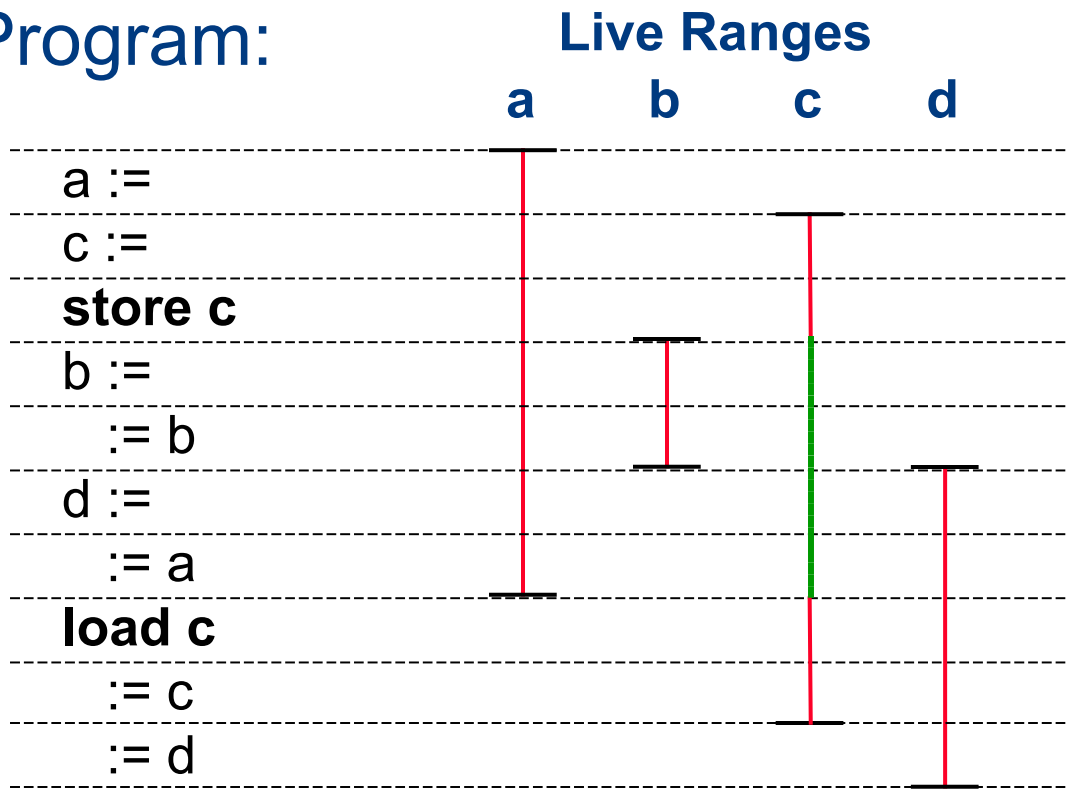
Register allocation using graph coloring

Spill/ Reload code

Spill/ Reload code is needed when there are not enough colors (registers) to color the interference graph

Example:
Only **two** registers available !!

Program:



Register allocation approaches

Method	Models Irregular Features	Fast	Optimal
Graph Coloring	✗	✓	✗
Integer Programming **[Goodwin and Wilken 96]** **[Kong and Wilken 98]** **[Fu and Wilken 2002]**	✓	✗	✓
Separated IP **[Appel and George 01]**	✓	✗	✗
PBQP **[Scholz and Eckstein 02]**	✓	✓ / ✗	✗ / ✓

Reading material

- Efficient DAG construction and heuristic calculation for instruction scheduling
  - Smotherman et al. (1991)
- Swing Modulo Scheduling: A liftime-sensitive approach
  - Llosa et al. (1996)

Both available on Oncourse (after the lecture)

Anouncement (1)

Deadline for assignment 1

- Full points if handed in before February 25th (next week Thursday)
- Half points if handed in after then

Assignment 2 starts Monday

- Instruction Level Parallelism estimation at the IR level

Anouncement (2)

Google summer of code

Spend your summer break writing code and learning about open source development while earning money! Accepted students work with a mentor and become a part of the open source community. Many become lifetime open source developers! The 2016 student application window is March 14th to 25th.

Technische Universiteit
Eindhoven
University of Technology