Phase Coupling Problems and Code Optimization by Integer Linear Programming

Overview

- Phase coupling problem of code generation phases.
- Impact of architectural design choices on code generation phases.
- Integer linear programming and phase-coupled code generation.
- PROPAN: A retargetable postpass optimization framework.

The Phase Coupling Problem during Code Generation

- Main subtasks of compiler backends:
  - code selection: mapping of IR statements to machine instructions of the target processor
  - register allocation: map variables and expressions to registers in order to minimize the number of memory references during program execution
  - register assignment: determine the physical register used to store a value that has been previously selected to reside in a register
  - instruction scheduling: reorder an instruction sequence in order to exploit instruction-level parallelism and minimize pipeline stalls
  - resource allocation: assign functional units and buses to operations

- Most of these tasks are interdependent, i.e. decisions made in one phase may turn out to be suboptimal in terms of overall code quality since they impose restrictions to other phases.

The Phase Coupling Problem in Code Generation

- Solution in a perfect world: Address all problems simultaneously in a single phase. BUT:
  - How to formulate this?
  - Code selection, register allocation/register assignment and instruction scheduling in general are NP-hard problems.
  - This means that in general there is no chance to optimally solve even one single of these tasks separately.

The Phase Coupling Problem in Code Generation

- Classical approaches: isolated solution by heuristic methods (list scheduling, trace scheduling, graph coloring register allocation, etc).

  $\Rightarrow$ Problem: interdependence of code generation phases.
  $\Rightarrow$ Suboptimal combination of suboptimal partial results.
  $\Rightarrow$ Inefficient code.

- Purely heuristic phase coupling (mutation scheduling, etc.) :
  - Code quality depends on the chosen heuristics to a large degree.
  - Efficiency of the heuristics depends on the target architecture.

  $\Rightarrow$ Goal conflict between easy retargetability and high code quality.

Code Selection and Register Allocation

- The goal of code selection is to determine the cheapest instruction sequence for a subgraph of the IR. However, the code selector does not know what the real overall costs of the instruction sequence will be; it can use only estimations.
- Register allocation usually is done after code selection so the code selector typically has to assume an infinite number of registers (virtual registers).
- In consequence when estimating the cost of an instruction sequence the code selector will mostly assume register references.
- Register allocation has to cope with a finite number of registers. If there are too few registers, spill code is generated.
- However, the cost of this spill code has not been considered during code selection, so the chosen operation sequence may in fact be a bad choice since another one might have worked without spill code.

Register Allocation and Instruction Scheduling

- Goal of register allocation: minimize number of registers.
  - Consequence: false dependences caused by register reuse, limiting instruction-level parallelism.

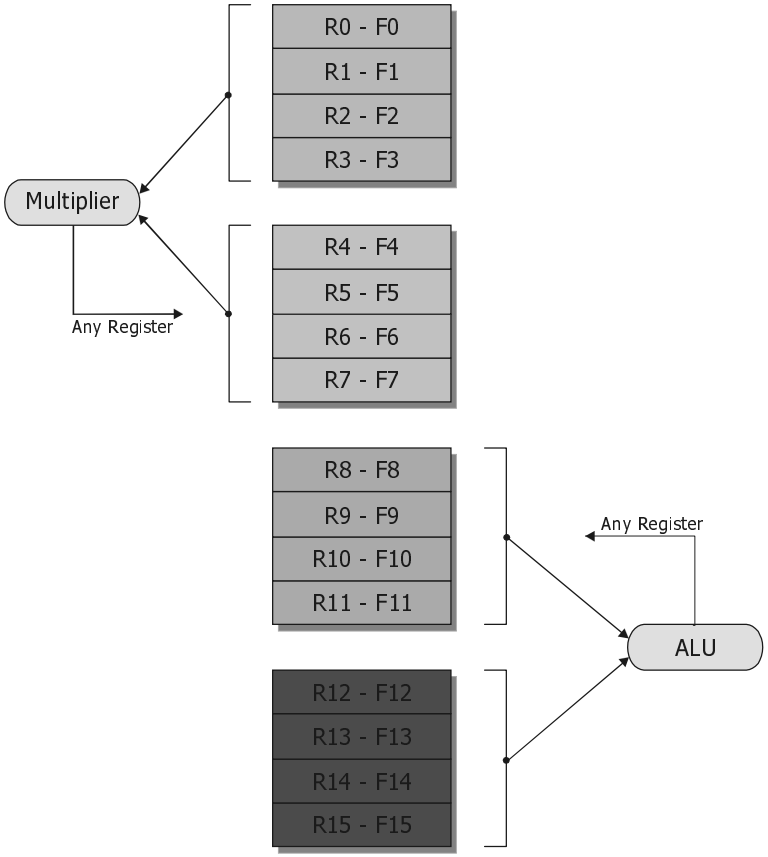
- Goal of instruction scheduling: maximize instruction-level parallelism.
  - Consequence: parallelization often forbits register reuse, possibly triggering generation of spill code during register allocation.

- Whichever task is executed first: it can make decisions which are globally suboptimal due to restrictions they impose to the second task.

  (Example for phase-coupling problem between code selection and instruction scheduling: later.)

Instruction Scheduling and Register Assignment

Analog Devices SHARC: Restricted Parallelism between ALU and multiplier.



R1=R1*R4
R2=R8+R12

```
R1 = R1 * R4, R2 = R8 + R12
```

```
R1 = R1 * R3
R2 = R8 + R12
```

```
R1 = R1 * R3
```

```
R2 = R8 + R12
```

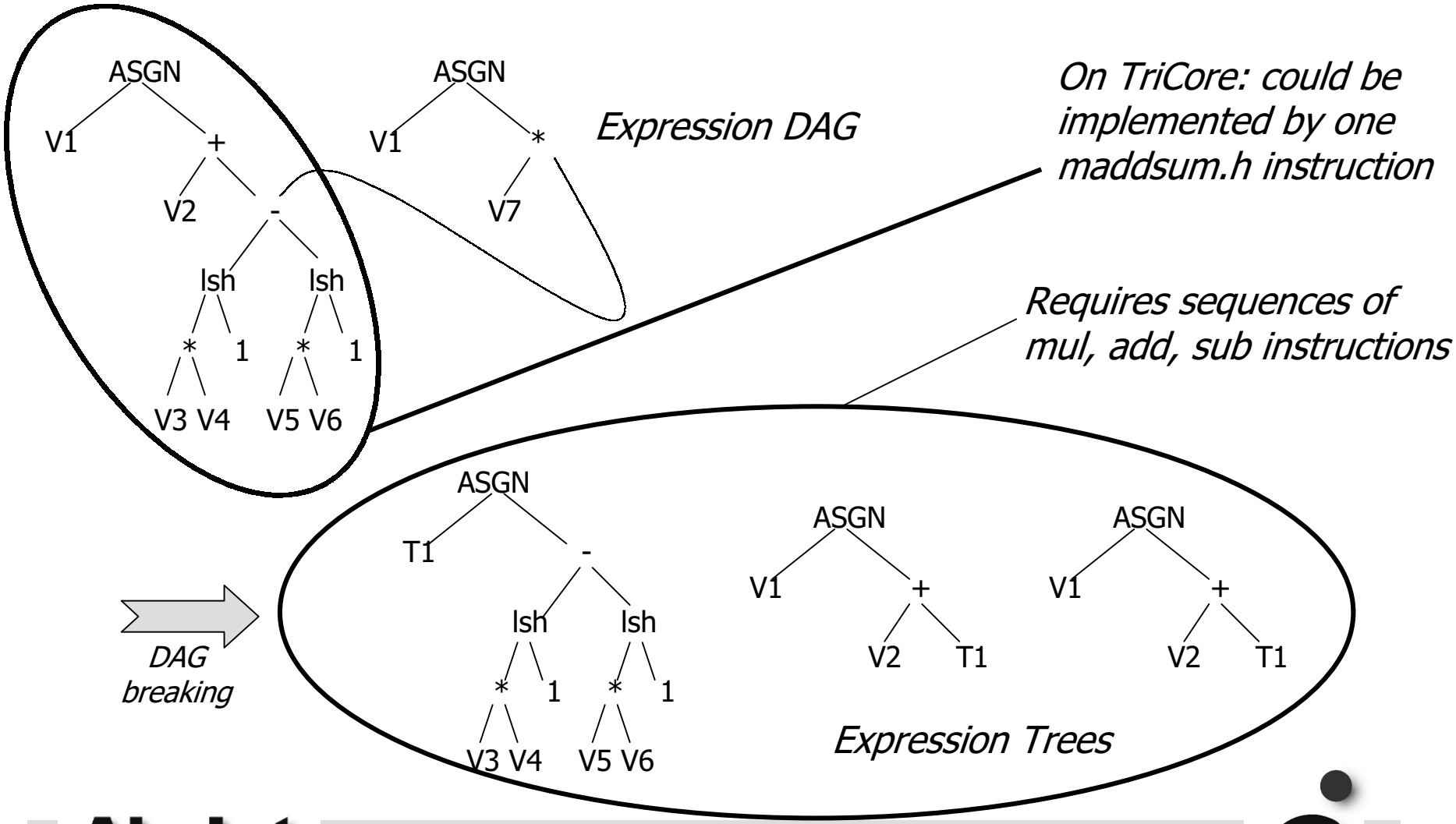
Overview

- ✓ Phase coupling problem of code generation phases.
- ✓ Impact of architectural design choices on code generation phases.
- Integer linear programming and phase-coupled code generation.
- PROPAN: A retargetable postpass optimization framework.

Code Selection and CISC Instructions

- Code selection usually is done by tree parsing (tree pattern matching) and dynamic programming.
- Usuall the IR however takes the form of directed acyclic graphs, e.g. due to common subexpressions.
- Before code selection proper DAGs must be broken into trees for the code selection algorithm to work.
- Breaking the trees is done heuristically. Thus the resulting trees and the resulting use of temporary storage locations may destroy the opportunity of generating complex instructions which would correspond to larger expression trees.

Code Selection and CISC Instructions



Code Selection and Complex Data Routes

- Complex data paths can have the consequence that moving a value from one register to another takes place across a route of register sets.
  - Code selection: Breaking the expression DAG into trees can lead to choosing a temporary storage location for one tree which leads to a sequence of register moves in other trees using this value.
  - Other consequence: the cost of one subtree depends on the code selected for other subtrees. This violates the precondition for dynamic programming which is commonly used to implement code selection.

Code Selection and Instruction-Level Parallelism

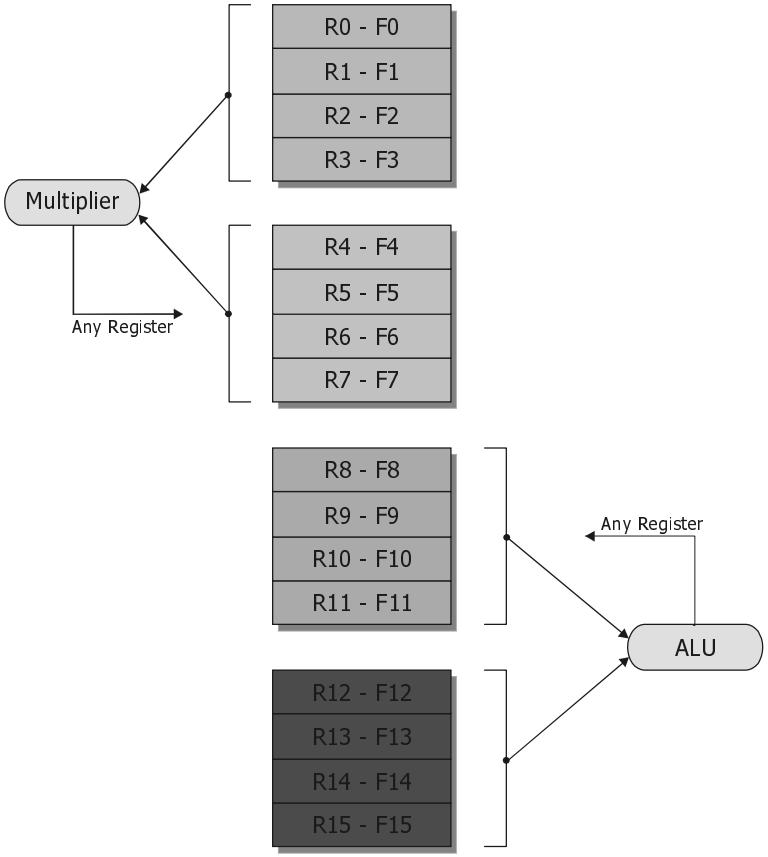
- For architectures exhibiting instruction-level parallelism operations from different expression trees can be executed in parallel if no dependences are violated.

- This cannot be taken into account during code selection, since the code of different subtrees is not independent any more. Again this violates the independence precondition of dynamic programming. A code sequence that locally seems clumsy may be the best one since it might allow many parallelizations with operations from other expression trees.

- Consequence: the quality of code selection can be severely affected due to the phase-coupling problem between instruction scheduling and code selection.

Heterogeneous register sets

Phase coupling problem between instruction scheduling and register assignment.



R1=R1*R4
R2=R8+R12

```
R1 = R1 * R4, R2 = R8 + R12
```

```
R1 = R1 * R3
R2 = R8 + R12
```

```
R1 = R1 * R3
```

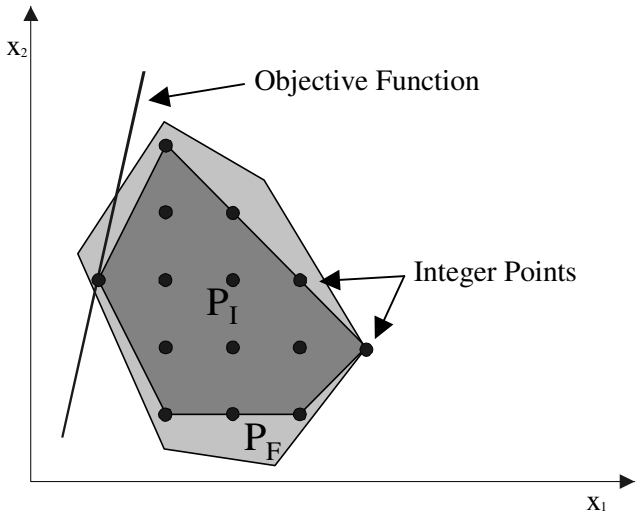
```
R2 = R8 + R12
```

Overview

- ✓ Phase coupling problem of code generation phases.
- ✓ Impact of architectural design choices on code generation phases.
- Integer linear programming and phase-coupled code generation.
- PROPAN: A retargetable postpass optimization framework.

Integer Linear Programming

- Integer linear programming is NP-complete, thus computing a provably optimal solution requires a structured formulation.
- $P_F=P_I$: Optimal integer solution can be computed in polynomial time.
- ILP allows to incorporate several code generation phases, e.g. instruction scheduling, register assignment and resource allocation (functional unit allocation) in a homogeneous problem description and to solve them optimally.



- Optimal phase-coupled code generation can be done for hot code sequences, ILP-based approximations provide high-quality approximative solutions.

ILP-Structure

$$\min \mathrm{M}_{\mathrm{steps}}$$

$$t_j \leq \mathrm{M}_{\mathrm{steps}} \qquad \forall\ j$$

Precedence Constraints

Assignment Constraints

Resource Constraints

ILP-Modelling Styles

- ILP-formulations in the area of code generation:
  - Time-indexed formulations: Choice of decision variables is based on the points of time the modeled events are assigned to.
    - $x_{jn}^{k} = 1$: operation *j* is started in clock cycle *n* by resource type *k*.
  - Order-indexed formulations: Decision variables reflect the ordering of the modeled events.
    - $x_{ij}^{k} = 1$: operation *j* is executed after *i* by resource type *k*.

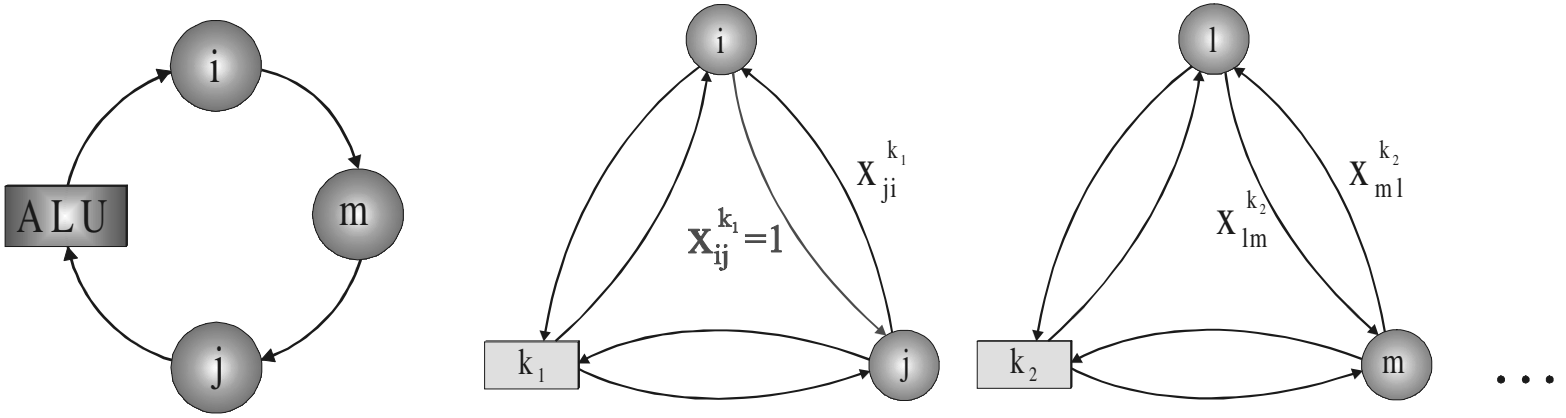
Code Generation By Integer Linear Programming

- Well-structured ILP formulations:
  - SILP (Scheduling and Allocating with Integer Linear Programming) [Zhang,96],[Kaestner,00]: order-indexed.
  - OASIC (Optimal Architectural Synthesis with Interface Constraints) [Gebotys,Elmasry,93],[Kaestner00]: time-indexed.

SILP	**OASIC**
Instruction scheduling	Instruction scheduling
Register assignment	Resource allocation
Resource allocation	Register assignment

The SILP Formulation (1)

- Order-based formulation.
- The main decision variables describe the flow of the hardware resources through the operations of the program: $x_{ij}^{k} \in \{0,1\}$
- The resource flow graph:



The SILP Formulation (2)

$$\min \mathrm{M}_{\mathrm{steps}}$$

$$t_j \le \mathrm{M}_{\mathrm{steps}} \qquad \forall j$$

$$t_j - t_i \ge w_i \qquad \forall (i,j) \in E_D^t$$

$$t_j - t_i \ge w_i - w_j + 1 \qquad \forall (i,j) \in E_D^o$$

$$t_i \le t_i + w_j - 1 \qquad \forall (i,j) \in E_D^a$$

Data Dependences

$$\sum_{(i,j)\in E_F^k} x_{ij}^k - \sum_{(j,m)\in E_F^k} x_{jm}^k = 0 \qquad \forall j \, \forall k$$

$$\sum_k \sum_{(i,j)\in E_F^k} x_{ij}^k = 1 \qquad \forall j$$

$$\sum_{(k,j)\in E_F^k} x_{kj}^g \le R_k \qquad \forall k$$

$$t_j - t_i \ge \mathrm{z}_i + \left(\sum_k x_{ij}^k - 1 \right) * U \qquad \forall (i,j) \in E_F^k$$

Flow Modelling

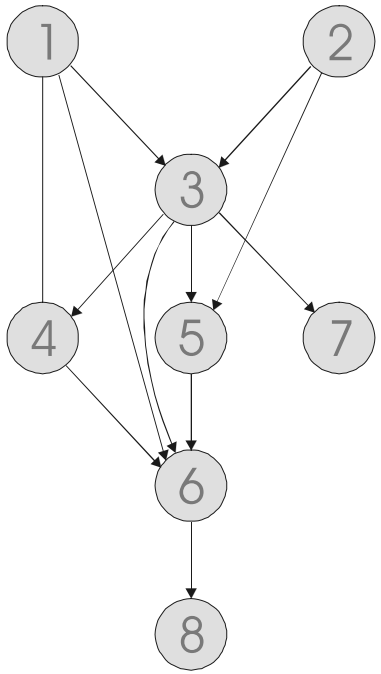
- Number of constraints: $O(n^2)$
- Number of variables: $O(n^2)$

Example (1)

```
1    r1 = M[...];
2    r2 = M[...];
3    r3 = r1 + r2;
4    r4 = r3 * r1;
5    r5 = r2 + r3;
6    r1 = r4 + r5;
7    M[...] = r3;
8    M[...] = r1;
```

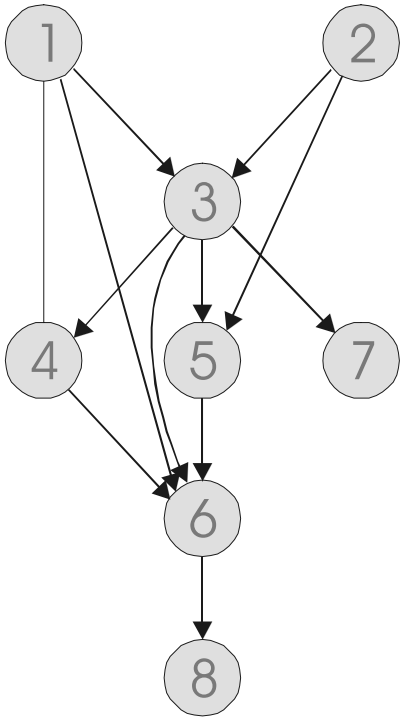
Functional Units:
- ALU:1
- MUL:1
- MEM:1

Execution Time: 1 clock cycle
Latency: 1 clock cycle



Data Dependence Graph

Precedence Constraints



$$t_j - t_i \geq w_i \qquad \forall \text{ true dependences } (i,j)^t$$

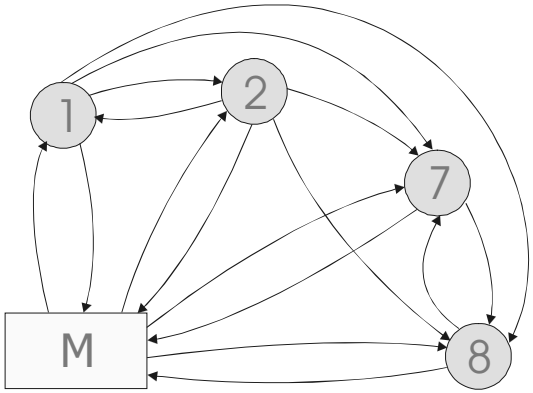
$$t_i \leq t_j + w_j - 1 \qquad \forall \text{ anti dependences } (i,j)^a$$

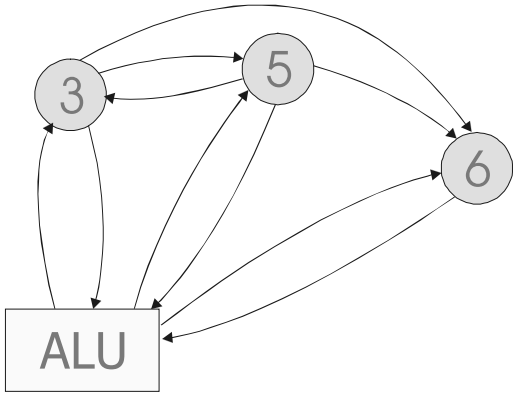
$$t_j - t_i \geq w_i - w_j + 1 \qquad \forall \text{ output dependences } (i,j)^o$$

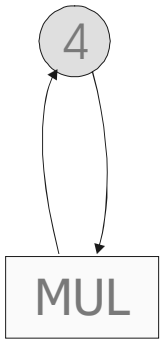
$$t_3 - t_1 \geq 1$$
$$t_3 - t_2 \geq 1$$
$$t_4 - t_3 \geq 1$$
$$t_5 - t_3 \geq 1$$
$$t_7 - t_3 \geq 1$$
$$t_6 - t_4 \geq 1$$

$$t_6 - t_5 \geq 1$$
$$t_6 - t_1 \geq 1$$
$$t_4 \leq t_6$$
$$t_3 \leq t_6$$
$$t_8 - t_6 \geq 1$$

Flow Conservation Constraints







$$x_{21}^{M} + x_{M1}^{M} - x_{12}^{M} - x_{17}^{M} - x_{18}^{M} - x_{1M}^{M} = 0$$

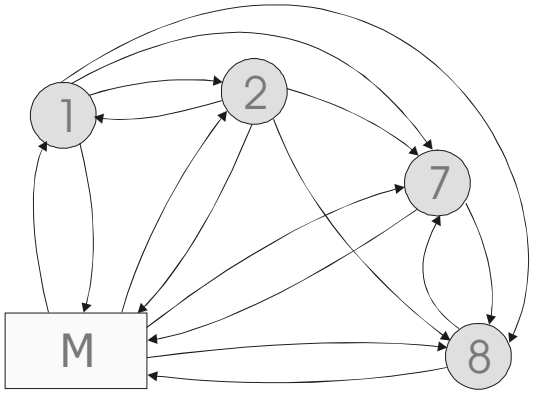
$$x_{12}^{M} + x_{M2}^{M} - x_{21}^{M} - x_{27}^{M} - x_{28}^{M} - x_{2M}^{M} = 0$$

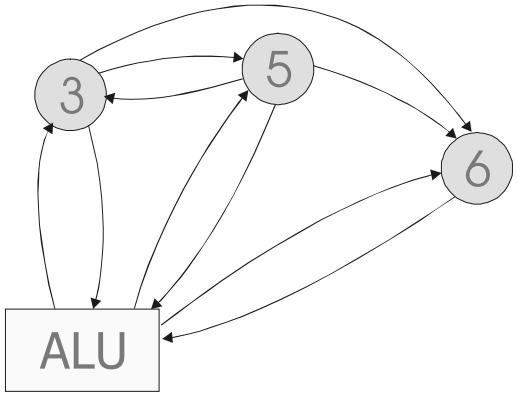
$$x_{17}^{M} + x_{27}^{M} + x_{87}^{M} + x_{M7}^{M} - x_{78}^{M} - x_{7M}^{M} = 0$$

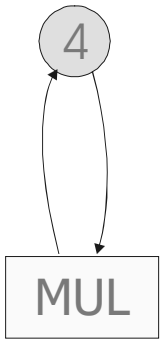
$$x_{18}^{M} + x_{28}^{M} + x_{78}^{M} + x_{M8}^{M} - x_{87}^{M} - x_{8M}^{M} = 0$$

...

Assignment Constraints







$$x_{21}^{M} + x_{M1}^{M} = 1$$

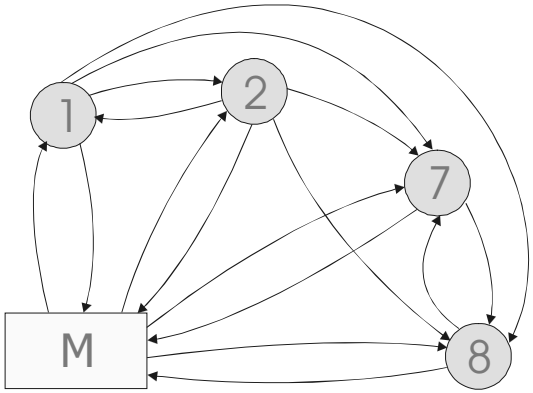
$$x_{12}^{M} + x_{M2}^{M} = 1$$

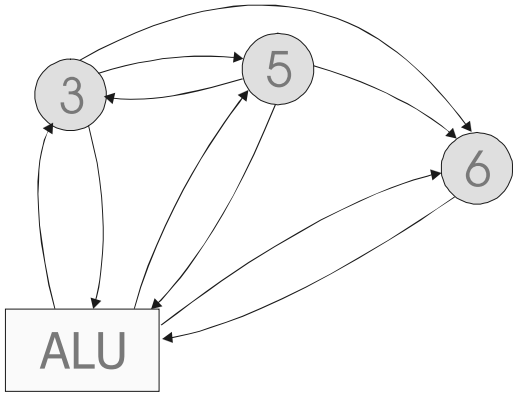
$$x_{17}^{M} + x_{27}^{M} + x_{87}^{M} + x_{M7}^{M} = 1$$

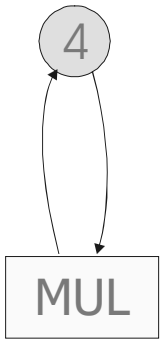
$$x_{18}^{M} + x_{28}^{M} + x_{78}^{M} + x_{M8}^{M} = 1$$

...

Resource Constraints





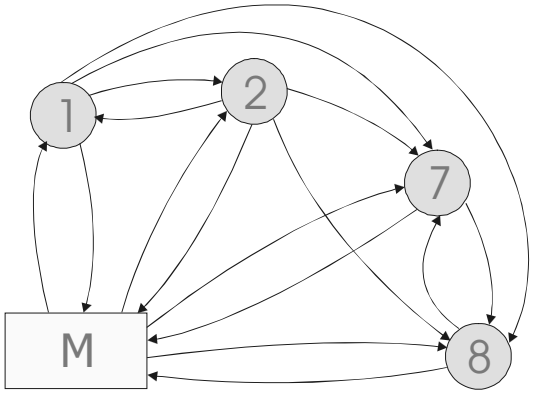


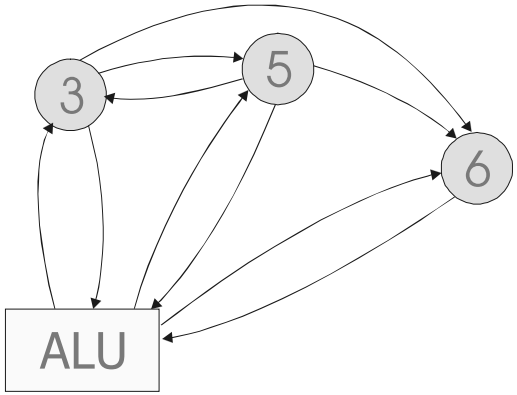
$$x^{M}_{M1} + x^{M}_{M2} + x^{M}_{M7} + x^{M}_{M8} \leq 1$$

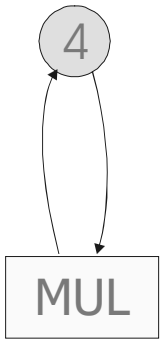
$$x^{Alu}_{Alu3} + x^{Alu}_{Alu5} + x^{Alu}_{Alu6} \leq 1$$

$$x^{Mul}_{Mul4} \leq 1$$

Serial Constraints







$$t_2 - t_1 \geq 1 + (x_{12}^M - 1) \cdot 8$$

$$t_1 - t_2 \geq 1 + (x_{21}^M - 1) \cdot 8$$

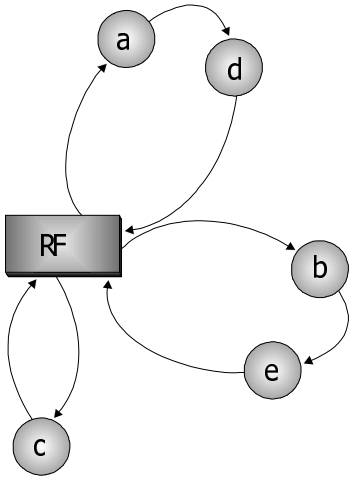
$$t_7 - t_1 \geq 1 + (x_{17}^M - 1) \cdot 8$$

...

Life Ranges and the Register Flow Graph

v_1 v_2 v_3 v_4 v_5

a) v_1 = **Mem[0xAFA0]**
b) v_2 = **Mem[0xAFC0]**
c) v_3 = v_1 + v_2
d) v_4 = v_1 * v_2
e) v_5 = v_3 + v_4
f) **return** v_5



a) R_1 = **Mem[0xAFA0]**
b) R_2 = **Mem[0xAFC0]**
c) R_3 = R_1 + R_2
d) R_1 = R_1 * R_2
e) R_2 = R_3 + R_1
f) **return** R_2

The Life Range Constraints

- t_i: starting time of operation i.
- w_i: execution time for operation i.
- τ_i: length of the life range of the variable defined by i.
- b_{ij}: distance between definition of a register by i and the use of that register by j.

- $t_j - t_i - b_{ij} = w_i \quad \forall$ true dependences (i, j)
- $\tau_i \geq w_i + b_{ij} \quad \forall$ true dependences starting from i

The Register Serial Constraints

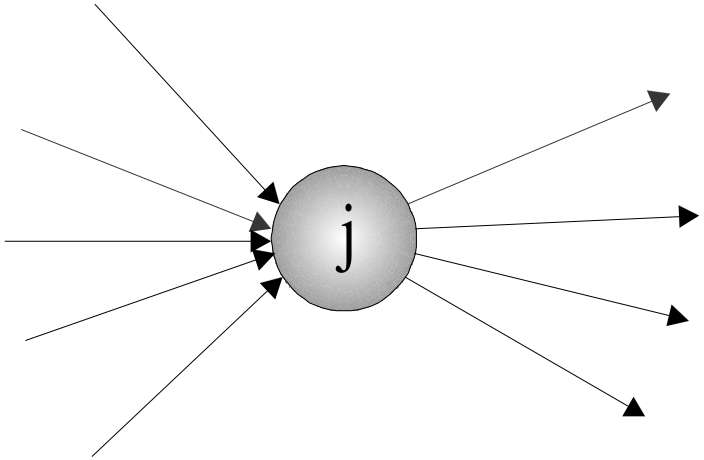
- U : upper bound for the execution time of the actual code sequence
- j may only write to the same register as a preceding instruction i if j is started after the life range of the variable defined by i has ended.
- serialization:

$$t_j - t_i \geq w_i - w_j + \tau_i$$

- $$t_j - t_i \geq w_i - w_j + \tau_i + \left(\sum_g x_{ij}^g - 1 \right) * U \quad \text{for all independent } i, j$$

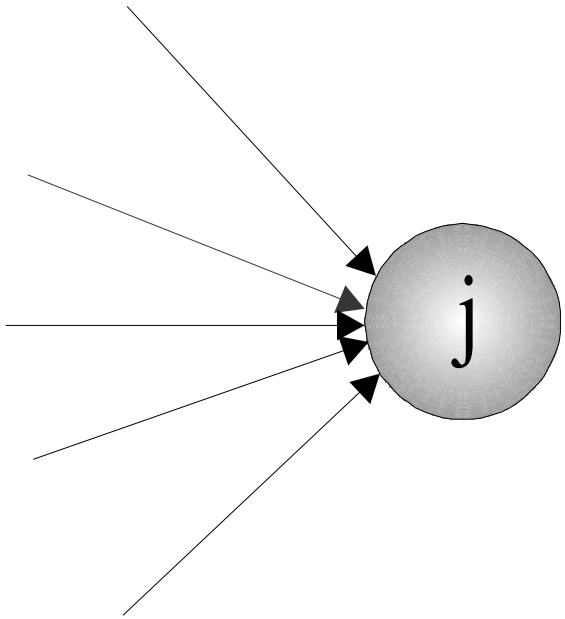
Flow Conservation Constraints

$$\sum_{(i,j)\in E_F^g} x_{ij}^g - \sum_{(j,m)\in E_F^g} x_{jm}^g = 0 \qquad \text{for all operations } j$$



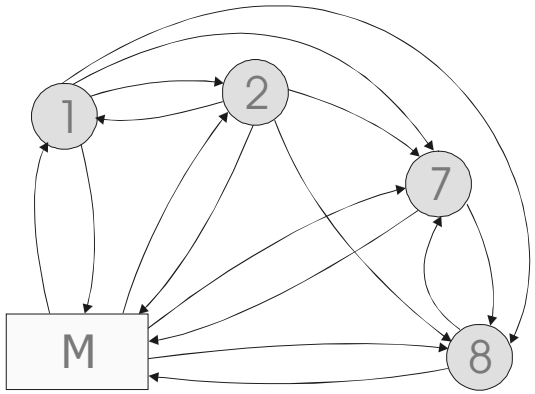
Register Assignment Constraints

$$\sum_{g} \sum_{(i,j) \in E_F^g} x_{ij}^g = 1 \quad \text{for all operations } j$$



Register Resource Constraints

$$\sum_{(g,j)\in E_F^g} x_{gj}^g \quad \leq \quad R_g \quad \text{for all register files } g$$



Overview of The Register Flow Constraints

$$t_j - t_i - b_{ij} = w_i \qquad \text{for all true dependences } (i, j)$$

$$\tau_i \geq w_i + b_{ij} \qquad \text{for all true dependences starting from } i$$

$$\sum_{(i,j)\in E_F^g} x_{ij}^g - \sum_{(j,m)\in E_F^g} x_{jm}^g = 0 \qquad \text{for all operations } j$$

$$\sum_g \sum_{(i,j)\in E_F^g} x_{ij}^g = 1 \qquad \text{for all operations } j$$

$$\sum_{(g,j)\in E_F^g} x_{gj}^g \leq R_g \qquad \text{for all register files } g$$

$$t_j - t_i \geq w_i - w_j + \tau_i + \left(\sum_g x_{ij}^g - 1 \right) * U \qquad \text{for all independent } i, j$$

- Number of constraints: $O(n^2)$
- Number of variables: $O(n^2)$

The OASIC Formulation (1)

- Time-based formulation.
- The main decision variables describe the assignment of an operation's starting time to a control step and a functional unit type.
- Main decision variables: x_{jn}^{k} where $x_{jn}^{k} = 1$ means that the execution of operation *j* is started in control step *n* on an instance of functional unit type *k*.

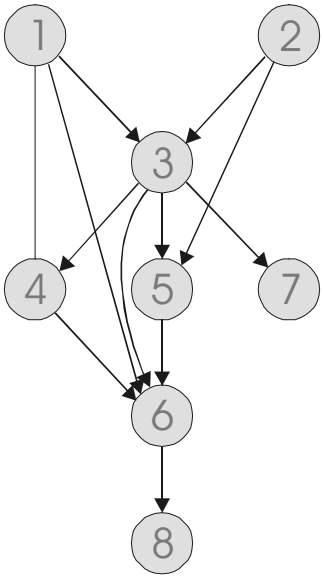
The OASIC Formulation (2)

$$\min \mathrm{M}_{\mathrm{steps}}$$

$$t_j = \sum_k \sum_{n=asap(j)}^{alap(j)} n \cdot x_{jn}^k \leq \mathrm{M}_{\mathrm{steps}} \qquad \forall\ j$$

- Precedence Constraints
- Assignment Constraints
- Resource Constraints
  - Number of constraints: $O(n^3)$
  - Number of variables: $O(n^2)$

Time & Precedence Constraints

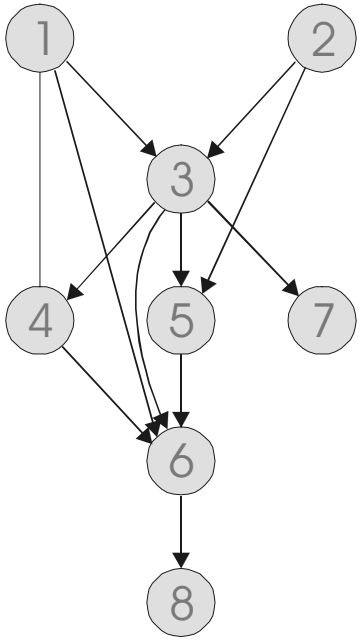


	ASAP	ALAP
1	1	4
2	1	4
3	2	5
4	3	6
5	3	6
6	4	7
7	3	8
8	5	8

$$t_1 = 1 \cdot x_{11}^M + 2 \cdot x_{12}^M + 3 \cdot x_{13}^M + 4 \cdot x_{14}^M \leq M_{steps}$$
$$t_2 = 1 \cdot x_{21}^M + 2 \cdot x_{22}^M + 3 \cdot x_{23}^M + 4 \cdot x_{24}^M \leq M_{steps}$$
$$t_3 = 2 \cdot x_{32}^M + 3 \cdot x_{33}^M + 4 \cdot x_{34}^M + 5 \cdot x_{35}^M \leq M_{steps}$$
$$\ldots$$

$$x_{12}^M + x_{32}^M \leq 1$$
$$x_{13}^M + x_{32}^M \leq 1$$
$$x_{13}^M + x_{33}^M \leq 1$$
$$x_{14}^M + x_{32}^M \leq 1$$
$$x_{14}^M + x_{33}^M \leq 1$$
$$x_{14}^M + x_{34}^M \leq 1$$

Assignment & Resource Constraints



	ASAP	ALAP
1	1	4
2	1	4
3	2	5
4	3	6
5	3	6
6	4	7
7	3	8
8	5	8

$$x_{11}^M + x_{12}^M + x_{13}^M + x_{14}^M = 1$$
$$x_{21}^M + x_{22}^M + x_{23}^M + x_{24}^M = 1$$
$$x_{32}^M + x_{33}^M + x_{34}^M + x_{35}^M = 1$$
$$\ldots$$

$$x_{11}^M + x_{21}^M \leq 1$$
$$x_{12}^M + x_{22}^M \leq 1$$
$$x_{13}^M + x_{23}^M + x_{73}^M \leq 1$$
$$\ldots$$