

Hardware-Software-Codesign

Felix Leitl

10. September 2025

Inhaltsverzeichnis

Architectures	3
General-Purpose Processors	3
Reasons for High Performance	3
General-Purpose Processors and Real-Time	4
Multimedia - Extensions and Sub-words	4
Microcontrollers	5
Digital Signal Processors	5
DSP - Arithmetics	6
DSP - Program Development	6
DSP - Trends	7
Application-specific instruction set processors (ASIP)	7
Integrated Circuit	7
Phases of IC Design	7
Design Styles for ICs	8
Comparison of Design Styles	8
FPGA (field programmable gate arrays)	9
FPGA Design Steps	9
FPGA - Application Areas	9
System models	10
Dataflow Graph (DFG)	10
Control flow graph (CFG)	10
Sequencing Graphs	11
Scheduling	13
Scheduling without resource constraints	13
Definition	13
Definition: Schedule	13
Definition: Latency	13
ASAP	13
ALAP	13
List scheduling	13
Periodic scheduling	14

Code generation	16
Double Roof	16
Compiler - Basics	16
Analysis	16
Synthesis	17
Syntax Tree abd DAG	18
3-Address Code	18
Basic Block	20
CFG and DAG for basic blocks	20
Code generation	20
Code (Instruction) Selection	21
Scheduling	21

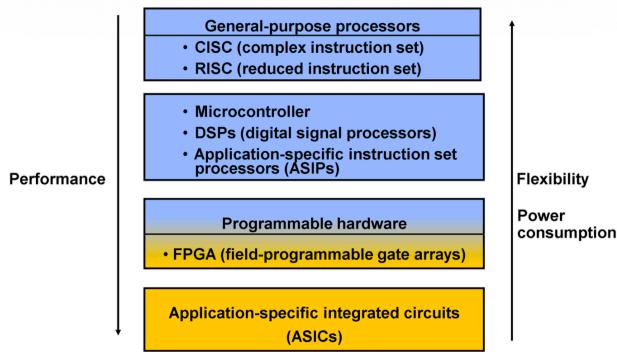


Abbildung 1: Implementation Styles

Architectures

See fig. 1

General-Purpose Processors

- Properties
 - high performance for a broad range of applications, not optimized for any particular application
 - high power consumption
- Design
 - highly optimized circuits
 - design times $\gtrsim 100$ person years
 - cost/device only cheap when fabricated in huge numbers
- Application areas
 - PCs
 - Smartphones

Reasons for High Performance

- Exploitation of parallelism
 - multiple scalar units (superscalar)
 - * Functional units: integer units, floating-point unit, load/store-unit
 - * Dynamic scheduling
 - * instruction compatibility

- * complex control unit
- deep instruction pipeline
 - * fetch | decode | read | execute | write back
 - * each scalar unit may have individual pipeline depth
 - * branch prediction
- Multi-level memory hierarchy
 - \uparrow Speed \Rightarrow \downarrow Size \Leftrightarrow \uparrow Size \Rightarrow \downarrow Speed
 - Register < Cache < Main Memory

General-Purpose Processors and Real-Time

- Execution time of programs is not well predictable
 - Dynamic scheduling
 - Caching
 - Branch prediction
- Complex I/O and memory interface

Multimedia - Extensions and Sub-words

- Multimedia Applications
 - 8 / 16 Bit data types
 - many arithmetic operations
 - huge data sets \Rightarrow Huge I/O-bandwidth
 - high data parallelism
- Sub-word execution
 - split 32/64-bit registers and ALUs into smaller sub-units
 - instructions execute in parallel on these sub-units
 - compromise between usage of parallelism and available data paths
 - currently most often based on hand-written assembly routines
 - Depends on language with defined bit width per data type and diverse overflow semantics
 - Compiler needs to automatically detect sub-word parallelism

Microcontrollers

- For control-dominant applications
 - control flow dominant programs (many branches and jumps)
 - few arithmetic operations
 - low throughput requirements
 - multitasking
- Microcontrollers are optimized for
 - Bit- and logic-level operations
 - Registers often realized in RAM: context switch through pointer operation, therefore shortest interrupt latencies
 - integrated peripheral units (e.g. A/D, D/A, CAN, Timer)
- Systems with 4/8 bit processors
- Microcontrollers with high performance
 - 16 - 64 bit processors
 - Application areas
 - * Systems with high control-dominant parts and additional demand
 - * High throughput (telecommunication, automotive)
 - * Computational power (industrial control, signal processing)
 - * As a part of an SOC

Digital Signal Processors

- Signal processing applications
 - data-flow dominant
 - many arithmetic operations, less branches and jumps
 - high degree of parallelism
 - high throughput requirements
- DSPs are optimized for
 - parallel instruction processing (MAC fig. 2)
 - Harvard Architecture, simultaneous access of multiple operands
 - zero-overhead loops
 - special addressing modes (circular, bit-revers)

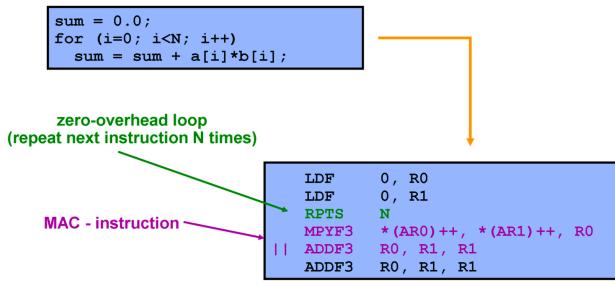


Abbildung 2: Multiply & accumulate

DSP - Arithmetics

- Number formats
 - Mantissa determines precision
 - Exponent determines dynamic range
- Fixed point
 - in case of equal mantissa length smaller (cheaper) and faster as floating-point
 - application design requires consideration of rounding and scaling problems
 - sufficient for many DSP applications
- Floating point
 - high dynamic range of numbers
 - easier application program development

DSP - Program Development

- Assembler, Complier
 - Many DSP features still only usable if code is written in assembly
 - Programming in C → profiling → time-critical parts in assembly code
- Libraries
 - Programming in C, call of hand-written optimized functions
- Code generation
 - Usage of design environments for modeling, simulation and code generation
 - e.g. Synopsis

DSP - Trends

- Multi-DSP Systems
 - for applications requiring highest performance
 - interfaces available to connect multiple processors
 - Multi-Processor SoC (MPSoC)
- VLIW (Very long instruction word)
 - multiple functional units
 - Compiler detects parallelism and creates a program that schedules multiple instructions jointly
 - Only useful for applications with high degree of instruction parallelism
 - suffers often from low code density
 - compilation difficult

Application-specific instruction set processors (ASIP)

- Specialization
 - instruction set (Operant chaining)
 - functional units (pixel operations, $1/\sqrt{x}$)
 - memory architecture (parallel access onto multiple memory banks)
- Advantages
 - higher performance
 - lower cost
 - smaller code images
 - lower power consumption

Integrated Circuit

Phases of IC Design

1. Design
 - (a) Modeling
 - (b) Synthesis & Optimization
 - (c) Verification
2. Fabrication
 - (a) Masks
 - (b) Wafer
3. Testing

4. Packaging

- (a) Slicing
- (b) Packaging

Design Styles for ICs

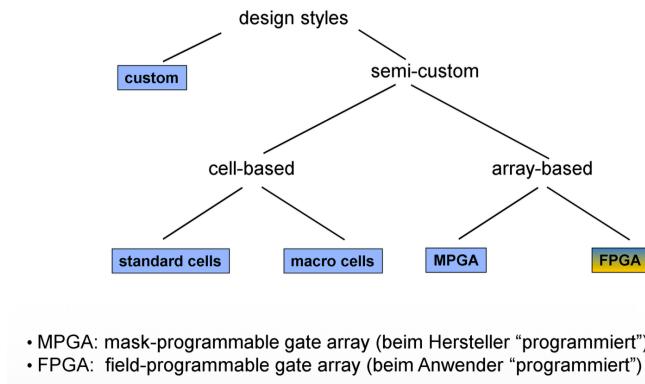


Abbildung 3: Design Styles for ICs

Comparison of Design Styles

	Custom	Cell-based	MPGA	FPGA
Density	very high	high	high	medium-low
Performance	very high	high	high	medium-low
Design time	very long	short	short	very short
Manufacturing time	medium	medium	short	very short
Cost-low volume	very high	high	high	low
Cost-high volume	low	low	low	high

Abbildung 4: Comparison of Design Styles

FPGA (field programmable gate arrays)

- Logic blocks
- I/O-blocks
- Interconnect

FPGA Design Steps

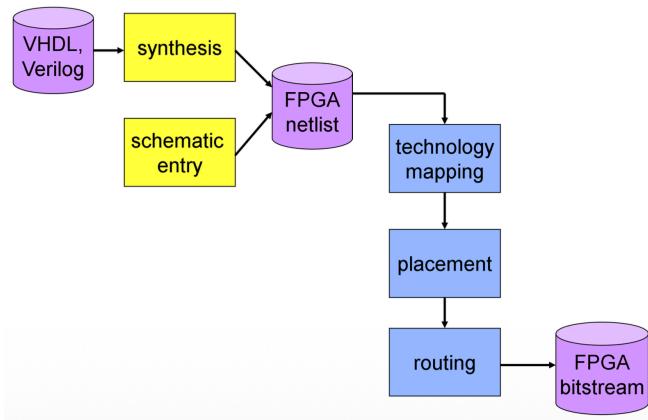


Abbildung 5: FPGA design steps

FPGA - Application Areas

- Glue Logic
- Rapid Prototyping, Emulation
- Embedded Systems
 - when processors are too slow or too power-inefficient and
 - * flexibility is a must
 - * the sale volumes are too low to afford ASIC
- Custom Computing
 - Goal: Combine flexibility of processors with efficient advantages of ASICs
 - e.g. Outsourcing a complex task from SW to HW

System models

Dataflow Graph (DFG)

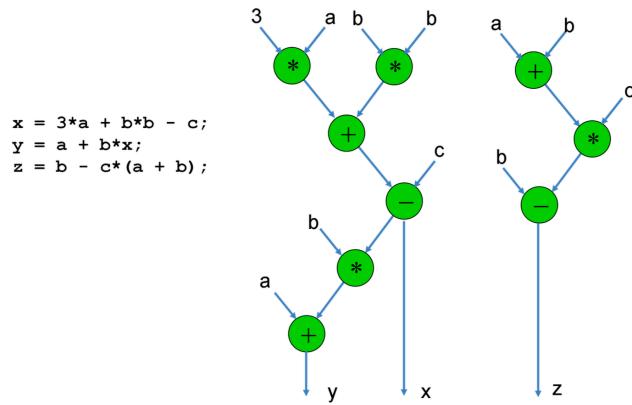


Abbildung 6: Data Flow Graph

Control flow graph (CFG)

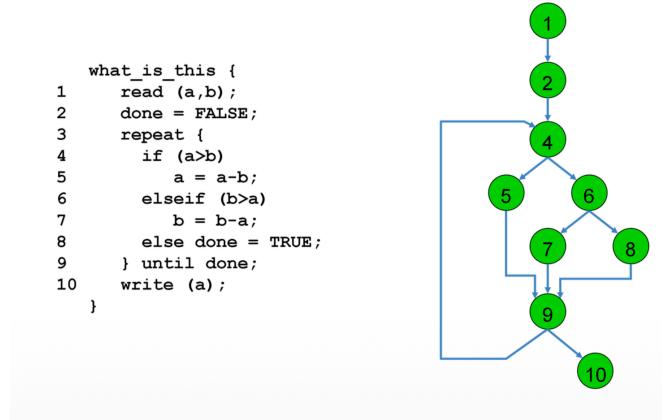


Abbildung 7: Control Flow Graph

Sequencing Graphs

- Hierarchy of entities
 - entities modeling the data flow
 - hierarchy modeling the control flow
- Special nodes
 - Start/End nodes: NOP
 - Branch nodes: BRANCH
 - Iteration nodes: LOOP
 - Module call nodes: CALL
- Attributes
 - Nodes: computation times, costs
 - Edges: conditions for branches, iterations

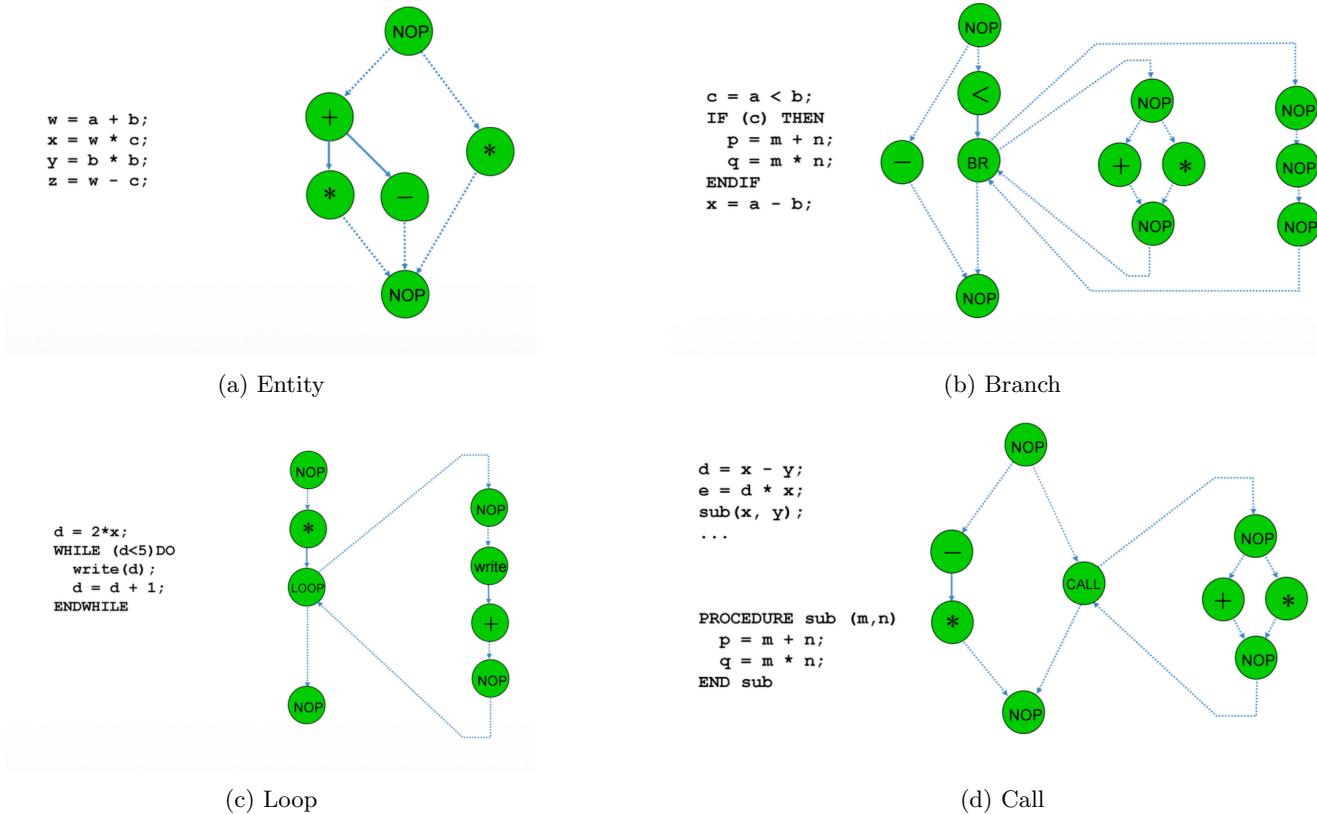


Abbildung 8: Sequencing Graph

Scheduling

- static vs. dynamic
- preemptive vs. non-preemptive
- with vs. without resource constraints
- aperiodic vs. periodic (iterative)

Scheduling without resource constraints

- ASAP (as soon as possible)
 - determines the earliest start times of tasks
 - computes the minimal latency
- ALAP (as late as possible)
 - determines the latest start times of tasks (for a given latency bound)
- Mobility of a task is given by the difference between ALAO and ASAP start times
 - Mobility 0 → task on critical path

Definition

Definition: Schedule

A schedule of a sequencing graph $G_S = (V_S, E_S)$ is a function $\tau : V_S \rightarrow \mathbb{N}$ that satisfies:

$$\tau(v_j) - \tau(v_i) \geq d_i \quad \forall (v_i, v_j) \in E_S$$

Definition: Latency

The latency L of a schedule τ of a sequencing graph $G_S = (V_S, E_S)$ is defined as:

$$L = \max_{v_i \in V_S} \{\tau(v_i) + d_i\} - \min_{v_j \in V_S} \{\tau(v_j)\}$$

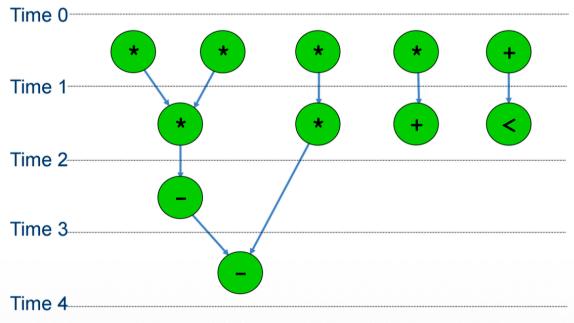
And therefore denotes the number of time steps of the smallest time interval that includes the execution of all nodes $v_i \in V_S$

ASAP

ALAP

List scheduling

- Tasks are sorted into a list according to some priority function
- In each step, each free resource will be assigned a schedulable task of highest priority
- Priority functions: number of successor nodes, mobility, ...



(a) latency-optimal schedule: $L = 4$

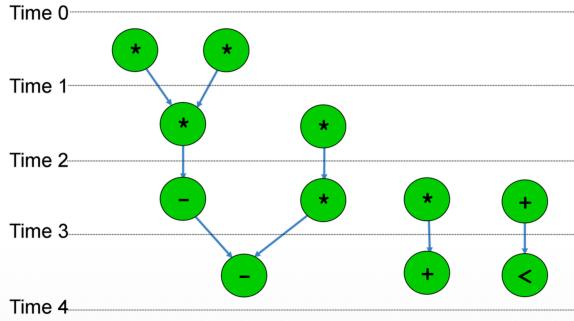
```

1: function ASAP( $G_S(V_S, E_S), d$ )
2:   for each  $v_i$  without predecessor do
3:      $\tau(v_i)^S \leftarrow 0$ 
4:   end for
5:   repeat
6:     Select a node  $v_i$  whose predecessors have all
      been scheduled
7:      $\tau(v_i)^S \leftarrow \max_{j:(v_j, v_i) \in E_S} \{\tau(v_j)^S + d_j\}$ 
8:   until all nodes  $v_i$  scheduled
9:   return  $\tau^S$ 
10: end function

```

(b) ASAP algorithm

Abbildung 9: ASAP



(a) latency bound: $\bar{L} = 4$

```

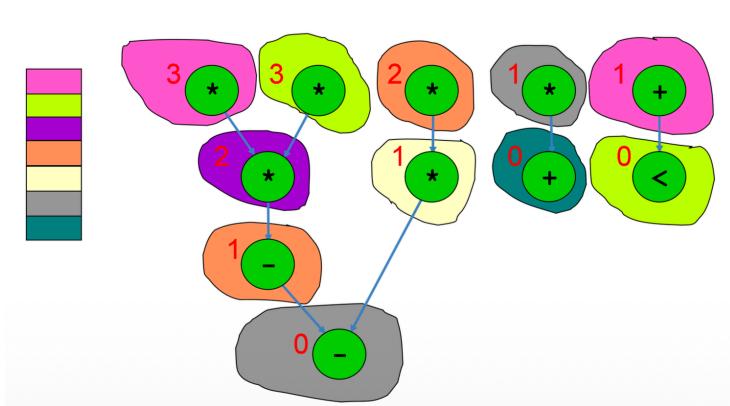
1: function ALAP( $G_S(V_S, E_S), d, \bar{L}$ )
2:   for each  $v_i$  without successor do
3:      $\tau(v_i)^L \leftarrow \bar{L} - d_i$ 
4:   end for
5:   repeat
6:     Select a node  $v_i$  whose successors have all
      been scheduled
7:      $\tau(v_i)^L \leftarrow \min_{j:(v_i, v_j) \in E_S} \{\tau(v_j)^L\} - d_i$ 
8:   until all nodes  $v_i$  scheduled
9:   return  $\tau^L$ 
10: end function

```

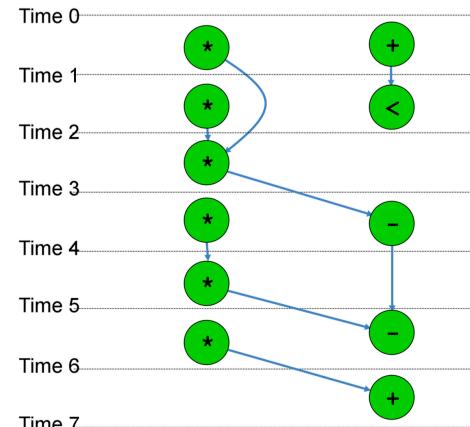
(b) ALAP algorithm

Abbildung 10: ALAP

Periodic scheduling



(a) Priority: Number of successor nodes
Resources: 1 multiplier, 1 ALU (+, -, <)



(b) Result

Abbildung 11: List schedule

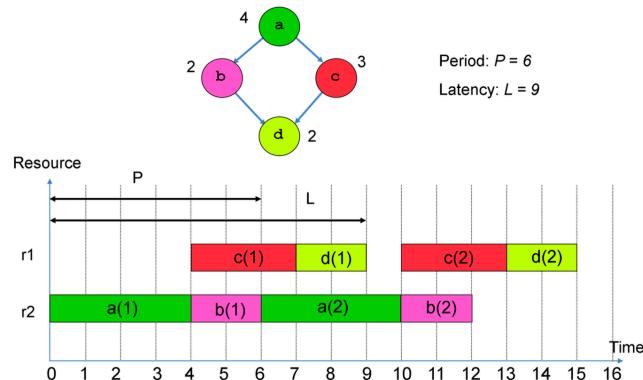


Abbildung 12: Periodic scheduling

Code generation

Double Roof

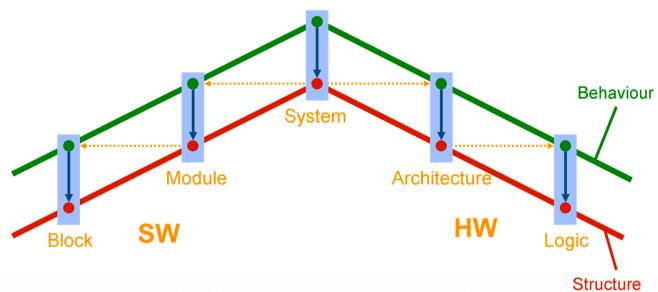
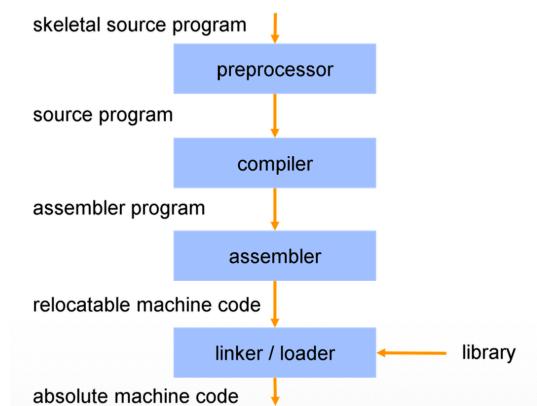
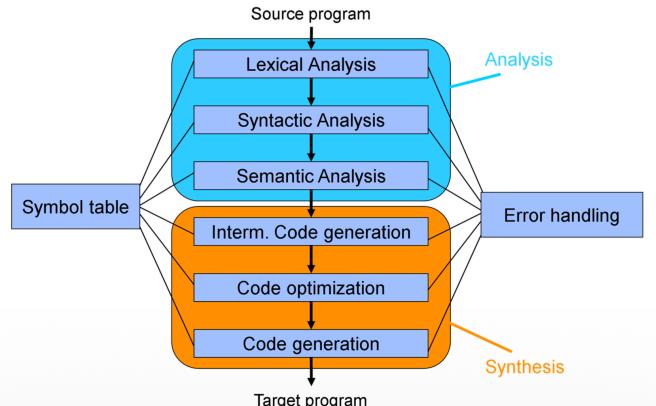


Abbildung 13: Double Roof

Compiler - Basics



(a) Compilation Process



(b) Compilation Process

Abbildung 14: Compiler

Analysis

- Lexical Analysis
 - scan source program and decompose it into symbols

- regular expression: recognition through finite automata
- syntactic analysis
 - parse sequences of symbols and determine clauses
 - clauses are described by a context free grammar
 - * $Z \rightarrow \text{Identifier} := A$
 - * $A \rightarrow A + A \mid A * A \mid \text{Identifier} \mid \text{Number}$
- semantic analysis
 - assure that the pieces fit together semantically
 - Example: type conversion

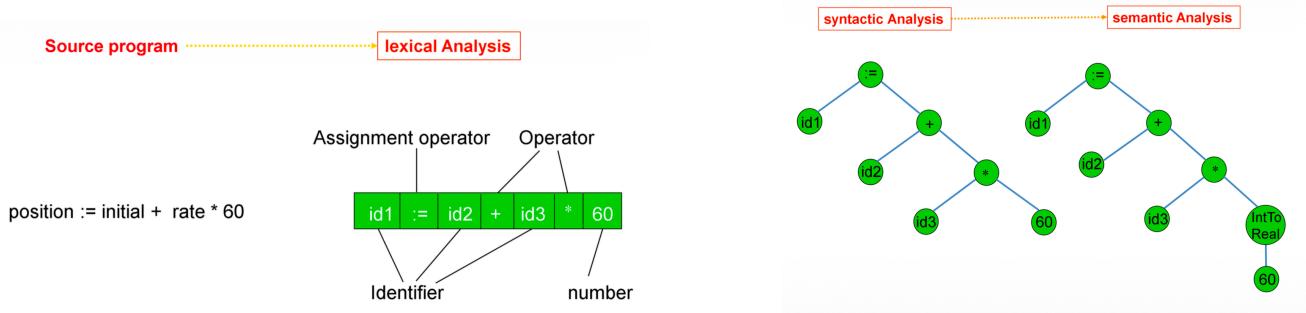


Abbildung 15: Analysis Example

Synthesis

- Generation of intermediate code
 - machine independent → retargeting easier
 - easy to generate
 - easy to be translated
- Optimization
 - General-purpose processors: fast code, fast translation
 - Special processors: fast code, compact code, small memory image
 - of intermediate code or of target code
- Code generation

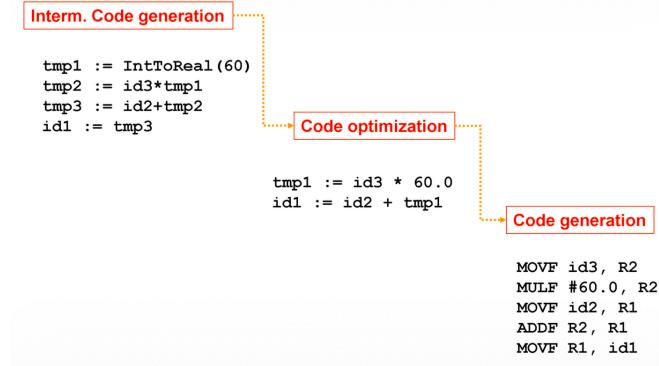


Abbildung 16: Synthesis Example

$a := b*(c-d) + e*(c-d)$

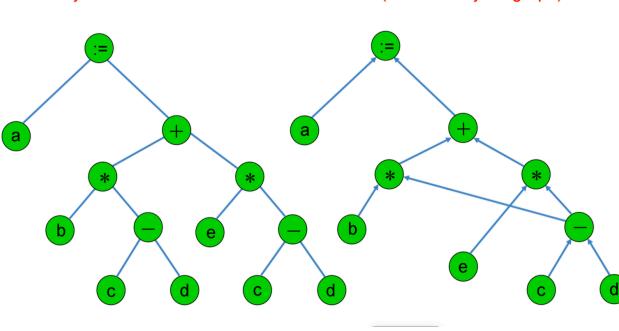


Abbildung 17: Syntax Tree and DAG

Syntax Tree abd DAG

3-Address Code

- Instructions
 - maximum 3 addresses (2 operands, 1 result)
 - maximum 2 operants
- Assignments
 - $x := y \text{ op } z$
 - $x := \text{op } y$
 - $x := y$
 - $x := y[i]$

- $x[i] := y$
- $x := &y$
- $y := *x$
- $*x := y$

- Control flow

- `goto L`
- `if x relop y goto L`

- Sub programs

- `param x`
- `call p,n`
- `return y`

- Advantages

- resolution of lengthy expressions and nested loops
- temporary names allow for easy reordering
- represents already a valid schedule

- Definition

- $x := y \text{ op } z$
- defines x and uses y and z

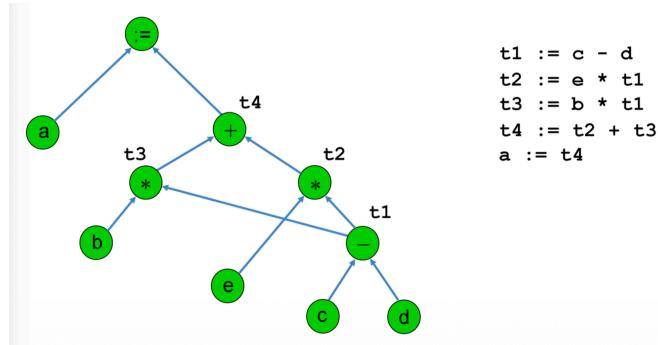


Abbildung 18: Generation of 3-address-code from DAG

Basic Block

Definition: A basic block is a sequence of consecutive instructions in which the control flow enters at the beginning and leaves at the end without branching except at the end.

Sequence of 3-address instructions → set of basic block:

- Determine block start points:

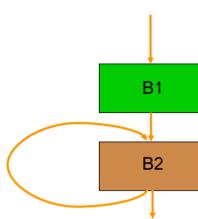
- the first instruction
- targets of branch and jump instructions
- instructions directly following a branch or jump instruction

- Determination of basic blocks

- each basic block includes its block start point
- includes all instructions (but excluding) the next block start point or until the program ends

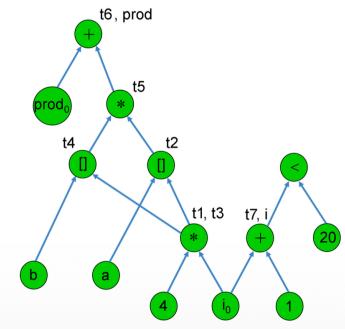
CFG and DAG for basic blocks

```
(1) prod := 0      B1
(2) i := 0
(3) t1 := 4 * i    B2
(4) t2 := a[t1]
(5) t3 := 4 * i
(6) t4 := b[t3]
(7) t5 := t2 * t4
(8) t6 := prod + t5
(9) prod := t6
(10) t7 := i + 1
(11) i := t7
(12) if i < 20 goto (3)
```



(a) Degenerated Control Flow Graph

```
t1 := 4 * i
t2 := a[t1]
t3 := 4 * i
t4 := b[t3]
t5 := t2 * t4
t6 := prod + t5
prod := t6
t7 := i + 1
i := t7
if i < 20 goto (3)
```



(b) Directed acyclic graph

Abbildung 19: CFG and DAG for basic blocks

Code generation

- Requirements
 - correct code
 - efficient code
 - efficient code generation
- Code generation = Software synthesis
 - Allocation: often given

- Binding:
 - * Register allocation, register binding
 - * Code (instruction) selection
- Scheduling
 - * Instruction sequencing
- Goal: efficient register usage
 - instructions on register operands typically shorter and faster than instructions with memory operands
- Register allocation, register binding
 - determine at each point of the program the set of variables that shall be stored in a register
 - bind each variable to a physical register
 - optimal register binding is an NP-complete problem
 - additional constraints given by special registers of the CPU architecture, compiler and operating system

Code (Instruction) Selection

- Code pattern for implementing a 3-address instruction [fig. 20]
- Problems
 - often inefficient code → Code optimization
 - There may be many alternative instructions
 - some instructions are only executable on certain registers
 - exploitation of special processor properties

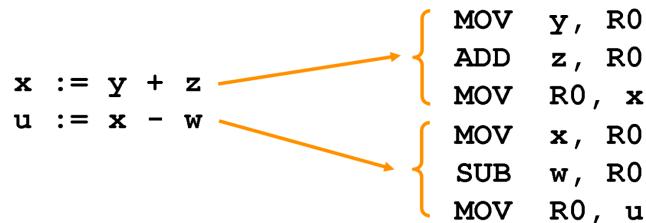


Abbildung 20: 3-address to assembler

Scheduling

Goal: efficient instruction execution sequences, as short as possible, using few registers

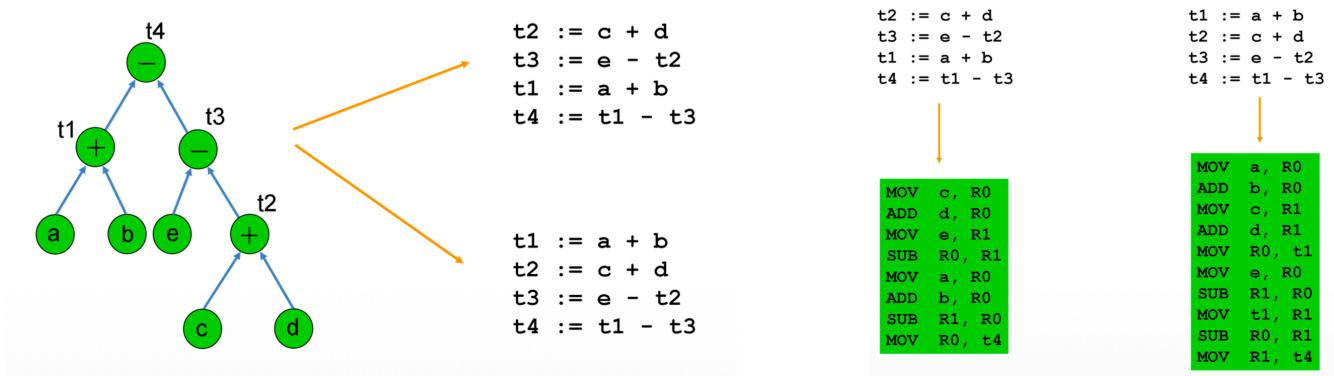


Abbildung 21: Scheduling differences