Department of Computer Engieneering

Hardware/Software Codesign I
Winter Term 2015/16

TECHNISCHE UNIVERSITÄT CHEMNITZ

Prof. Dr. Wolfram Hardt, Dipl.-Inf. Michael Nagler

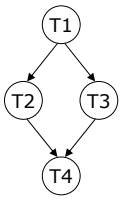
Informatik

Lesson 2

SYNTHESIS, SYSTEM MODELS

Exercise 1

The following system specification with 4 tasks $T1, ..., T4$ is given:



The tasks can be executed on different hardware units. The following table shows these units and the according costs and execution times.

		exec. time (ms)			
unit	cost/unit	T1	T2	T3	T4
MIPS	200	5	-	-	2
DSP	120	-	20	18	5
FPGA	240	-	12	10	-
ASIC	400	-	-	1	-

(a) Assume that every hardware unit can be used at most 1 time. Note the design space by listing all possible *Design Points*. A design point is a set of allocation, binding and scheduling.

(b) Determine the costs and the execution times of all design points.

(c) Draw the design point to a cost / execution-time diagram and mark the *Pareto Points*.

(d) Are there new design points, if each hardware unit can be used more than just 1 time?

(e) How many design points exist at least, if a task graph with n tasks is arbitrary mapped to a system with at least 3 hardware units?

a) b)

T_1	T_2	T_3	T_4	cost	Exec. time
M	D	D	M	320	5+20+18+2
M	D	D	D	320	5+20+18+5
M	D	F	M	560	5+20+2
M	D	F	D	560	5+20+5
M	F	D	M	–	25
M	F	D	D	–	28
M	R	F	M	440	27
M	F	F	D	680	32
M	D	A	M	720	27
M	D	A	D	720	30
M	F	A	M	840	18
M	R	A	D	860	22

c)

x_1 = cost
x_2 = Runtime

$x \in R^n$

$\exists y \in R^n$ with $y_i \in x_i$; $1 \le i \le n$ at least 1 point inequality

摘自 Lecture 中 Pareto Point 图

d) M F_1 $\bar{F}_2$ M 680 5+12+2

e) k^n k = objects
n = repititions (tasks)

Exercise 2

Graphs are a very important formalism for system modeling and automated synthesis.

(a) Give the definition of a *graph*.

(b) What are differences of directed and undirected graphs?

(c) A completely connected graph G is given with n nodes. How many edges exist in G for the cases that it is a directed or an undirected graph?

Exercise 3

Given is a set of tasks $T = \{A, B, C, D, E\}$ and the dependency relation $R = \{(A, B), (A, C), (C, D), (D, E), (C, E)\}$. Hereby, a relation (A, B) means that task A produces data for task B and thus has to be executed before B.

Furthermore, a micro controller (MC), a digital signal processor (DSP) and a configurable gate array (FPGA) are available to implement the system. All components are connected to a common system-bus and can communicate pairwise by bidirectional point-to-point connections.

Task A can only be implemented on the FPGA or on the DSP. Task E can only be implemented on the FPGA. All other tasks can run on all components.

(a) How is a *Specification Graph* defined?

(b) Note the specification graph for the mentioned system.

$G = (V, E)$

V = set of vertices / nodes

$E \subseteq V \times V : (u,v) \mid u,v \in V, u \neq v$ (Edges)

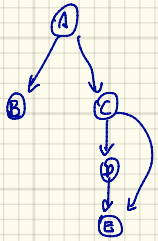
$\forall e_1, e_2 \in E \quad e_1 = (u,v) \quad e_2 = (v,u) \rightarrow e_1 = e_2 \rightarrow$ undirect Graph

$e_1 \neq e_2 \rightarrow$ directed Graph

Exercise 2

dir $\quad n \cdot n-1$

undirected $\quad \frac{n \cdot (n-1)}{2}$



specification graph

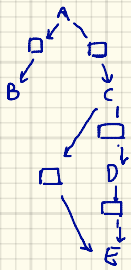
Problem graph — possible mapping — Architecture graph

V Task & Communication

E Dependencies

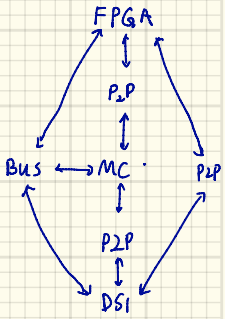
V HW components & comm. interface

E Direct con path



Exp. A → MC

And on each different path have different weight.



Department of Computer Engieneering

Hardware/Software Codesign I
Winter Term 2015/16

TECHNISCHE UNIVERSITÄT CHEMNITZ

Prof. Dr. Wolfram Hardt, Dipl.-Inf. Michael Nagler

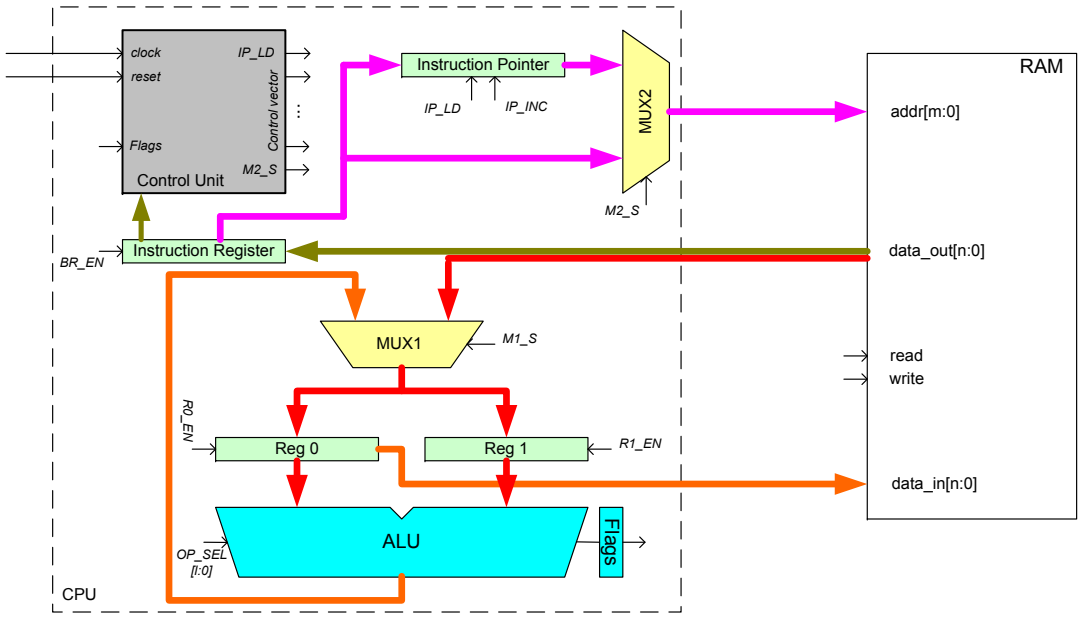
Informatik

Lesson 3

TARGET ARCHITECTURES (PROCESSORS)

Exercise 1

The following schematic of a simple *central processing unit* (CPU) is given:



(a) Of what kind of architecture is this processor? von – neiman

(b) Note an appropriate state machine for the control unit.

https://software.intel.com/zh-cn/articles/book-Processor-Architecture_CPU_work_process

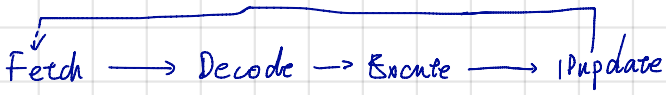
(c) What are the most important classes of commands for this processor? How are they executed by the CPU?

(d) What are disadvantages of this proposed architecture?

read data from RAM
Save data directly

Exercise 2

Compare important characteristics of CISC- and RISC-architectures.



CISC

complex Operator

memory management

different ex. time for each command

short program (less memory)

special command

complex compilers

RISC

only Res. Operator

fixed exec. time for each command

long programms

Pipelining

Exercise 3

(a) What are possible extensions for general purpose processors that accelerate the execution of multmedia applications.

(b) What is the meaning of a permutation command?

(c) Note a HW-implementation (Register-Transfer level) for the permutation command based on a 32-bit register with a subword size of 8 bit.

Exercise 4

Complete the following table to compare characteristics of a *microcontroller* and a *digital signal processor*:

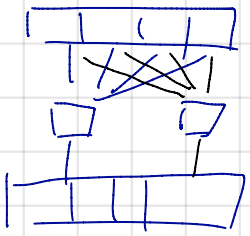
	Microcontroller	DSP
code-majority		
nr. of jumps/ branches		
nr. of bit-/ logic operations		
nr. of arithmetical operations		
data throughput		
peripheral devices		
parallelism		
real time properties		

Exercise 3

(a) special Induction sets (MMX.)

(b) sub-word execution

(c)



Exercise 4

MC	DSP
— control flow	— dataflow
many J/b	Less Jumper/branches
many b/logic operations	Less
Less arit op.	many
low	high
integraced	external
multitask	dedicated comm. (MAC)
Possible (but not typical for App)	possible

efficience

Exercise 5

The following equation is given:

$$y = \sum_{k=0}^{N-1} (c[k] * x[k])$$

(a) Note a C-like program, that calculates y.

(b) Note an assembler DSP program, that calculates y. You can use the following assembler commands:

Mnemonic	Parameter	Description
LDI	*value*, *reg*	Loads an integer *value* in a register *reg*
ADDI3	*reg_op*1, *reg_op*2, *reg_val*	Adds 2 register values *reg_op*1 and *reg_op*2 and stores the result in *reg_val*
MPYI3	*reg_op*1,*reg_op*2, *reg_val*	Multiplies 2 register values *reg_op*1 and *reg_op*2 and stores the result in *reg_val*
RPTS	*value*	Repeats the adjacent command *value*-times
OP1 ‖ OP2	jeweils *reg_op*1, *reg_op*2, *reg_val*	Executes the operations OP1 and OP2 in parallel (OP has to be MPYI3 or ADDI3)

To access the operands in the memory, two address registers $AR0$ and $AR1$ are available. They can be used like normal registers. At startup time $AR0$ points to the first value of c and $AR1$ points to the first value of x.

The value in the memory address of ARx can be accessed with $*ARx$.

The operation '$ARx++$' increments the address automatically after the access to the memory.

Int y =0;
For (int k=0; k<N ; k++)
{ y+= c[k] * x[k] }

```
LDI   0.R0   //multiplication
LDI   0,R1   // addition

RPTS  N
      ADDI3  R0, R1, R1
      MPY 3  *AR0++, *AR1++, R0
      ADDI3  R0, R1, R1
```

Department of Computer Engieneering

Hardware/Software Codesign I
Winter Term 2015/16

TECHNISCHE UNIVERSITÄT CHEMNITZ

Prof. Dr. Wolfram Hardt, Dipl.-Inf. Michael Nagler

Informatik

Lesson 4

Target Architectures (FPGA, ASIC)

Exercise 1

The following interface (ENTITY) and behavioral description (ARCHITECTURE) is given in the hardware description language VHDL:

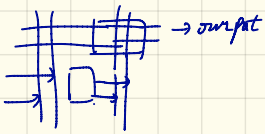
```
1   ENTITY ex1 is
2       PORT(    a:      IN BIT;
3                b:      IN BIT;
4                c:      IN BIT;
5                d:      IN BIT;
6                y:      OUT BIT;
7                z:      OUT BIT
8           );
9   END ex1;
10
11  ARCHITECTURE behavioral OF ex1 IS
12  SIGNAL      temp:   BIT;
13  BEGIN
14
15      temp    <= a AND b;
16
17      y       <= temp AND (c OR d);
18      z       <= '1' WHEN temp = '1';
19
20  END behavioral;
```

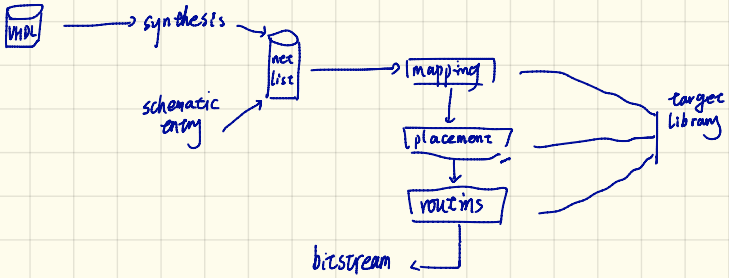
(a) Explain the structure of a *Field Programmable Gate Arrays* (FPGA).

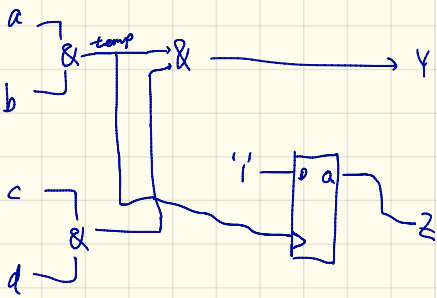
(b) Explain the different design steps to get from a given behavioral description to a FPGA programming file (bitstream).

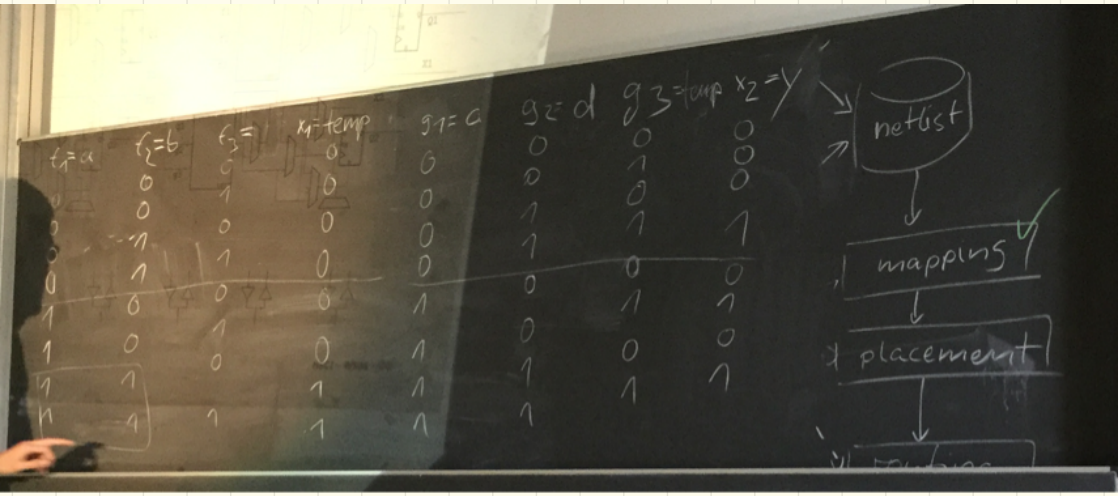
(c) In the figure at the end of this sheet, the schematic of a very simple FGPA is shown. Due to simplification, the interconnection lines are not shown. Explain how this FPGA works.

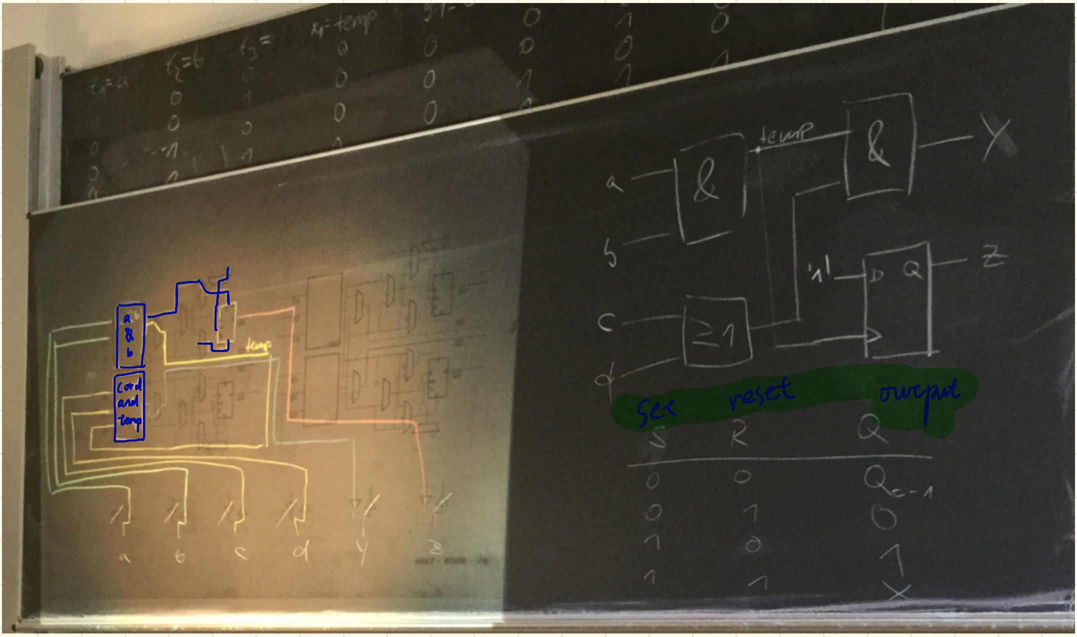
(d) Implement the behavioral description (given above) as a complete FPGA configuration for the device in the mentioned figure. Show all interim results from the mentioned steps of task (b).


output


VHDL
synthesis
schematic entry
net list
mapping
placement
routing
target Library
bitstream


a
b
c
d
&
temp
Y
'1'
D
Q
Z


netlist
mapping
placement


a & b
cond and temp
temp
a
b
c
d
Y
&
≥1
temp
'1'
D Q
Y
Z
set
reset
output
S R Q
0 0 Q_{t-1}
0 1 0
1 0 1
1 1 X

Exercise 2

Compare advantages and disadvantages of *Application Specific Integrated Circuits* (ASICs) and FPGAs.

Exercise 3

(a) Explain what is meant with *block-oriented design*?

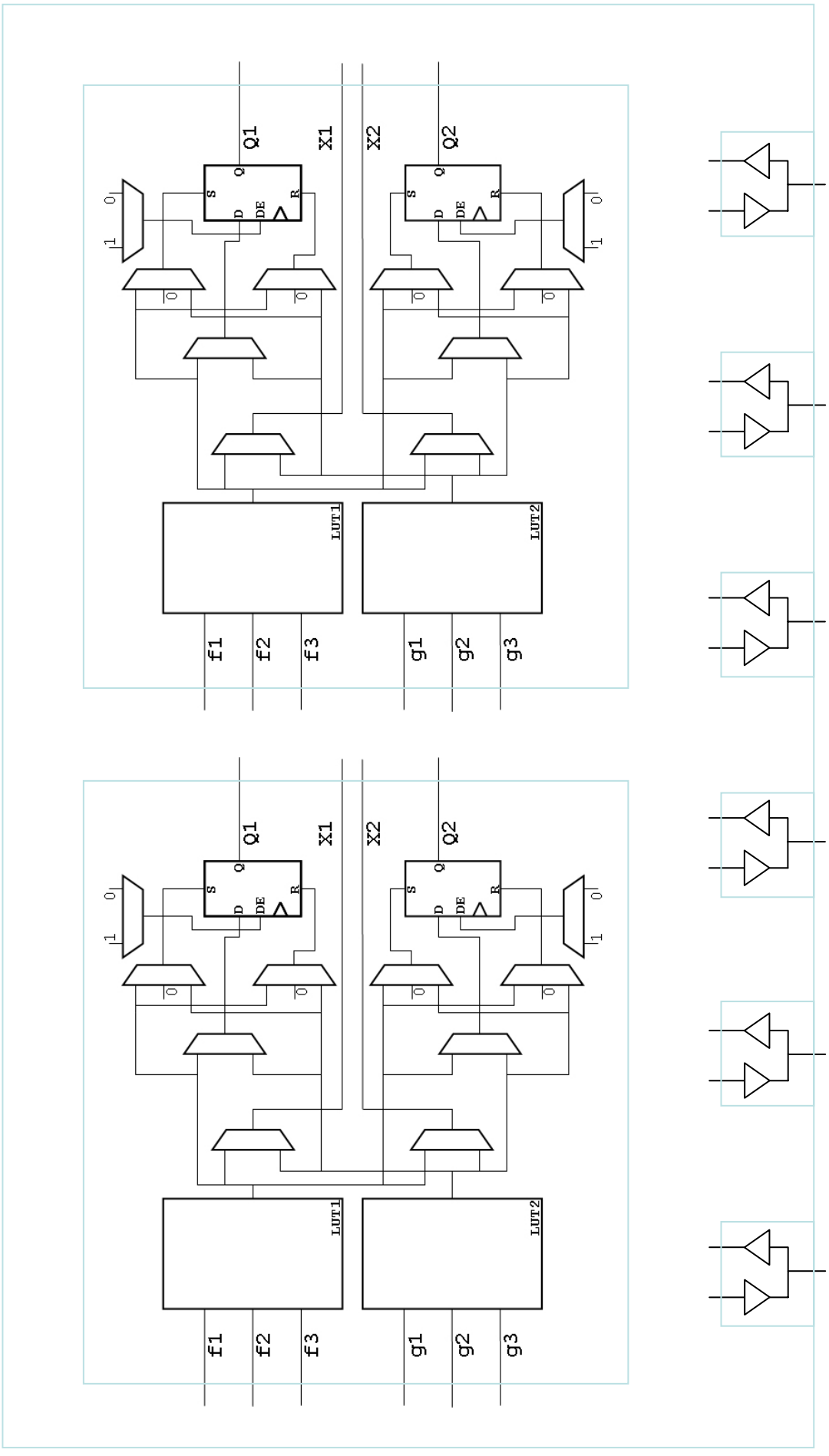
(b) Classify block types depending on their abstraction level. — on lecture ppt

(c) Blocks can be offered as *Intellectual Properties* (IPs). Name some important characteristics for the classification of IPs.

	Asic	FPGA
density	very high	medium low
performance	very high	medium to low
Dev. time (Development)	very long	short
production time	medium	short
cost – low amount – high amount	high low	low high

soft blocks
firm blocks
hard blocks

– cost
– flexibility / size
– timing / delays
– interfaces


Q1
X1
X2
Q2
S
D
DE
R
Q
LUT1
LUT2
f1
f2
f3
g1
g2
g3

Department of Computer Engieneering

Hardware/Software Codesign I
Winter Term 2015/16

TECHNISCHE UNIVERSITÄT CHEMNITZ

Prof. Dr. Wolfram Hardt, Dipl.-Inf. Michael Nagler

Informatik

Lesson 5

COMPILER

Exercise 1

What are the main phases and tasks of a *Compiler*?

Exercise 2

The following equation is given:

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

(a) Convert this equation to a *Syntax-Tree*.

(b) Convert this equation to a *Directed Acyclic Graph* (DAG).

(c) Convert this equation to *3-Address-Code*.

Exercise 3

(a) How is a *Basic Block* defined?

(b) Note an algorithm (C-similar notation) that determines the basic blocks of a series of 3-Address-Commands. The commands are given in an array. Furthermore, following functions can be used:

```
1  bool JumpCommand(codeline cl);
2       // true, if cl is a jump
3
4  int  JumpTarget(codeline cl);
5       // gives the target (as code line number)
6       // of a jump in cl
```

The algorithm has to output the assignment of each command to a basic block.

Lesson 5 Exper. 1

phases : analysis

source program → lexical. a. → words

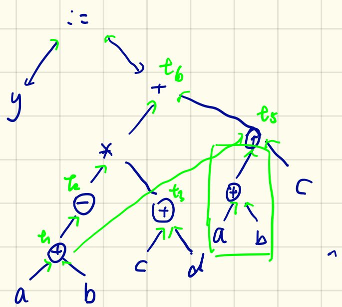
syntactic. a. → fullfill the sentence

semantic. a. → int a = 3.4 ; wrong

| intermidate code
| code optimsation
| code generation.

synthesis

Ex 2:



Green means DAG

- max 3 Addresses (2 operands, 1 result)
- max 1 operators and 2 oprands

$t_1 = a + b$

$t_2 = -t_1$

$t_3 = c + d$

$t_4 = t_2 * t_3$

$t_5 = t_1 + c$

$t_6 = t_4 + t_5$

$y = t_6$

Ex 3.

Basic Blocks 只有一个出入口且 执行不被中断 slide 4 Page 16

Beginning of BB

1. first instr. of code
2. introduction of func
3. target of jump.

```
Codeline [n];
bool jumpcommand(codeline c[])
int jumptarget(codeline c[])
int BB[n]                                   BB[0] = 1
// at least 2 iniraters the jmp target may      | 2
// before the jmp instruction)                  3
                                            [0]  3

int JT[n]
   for(int i=0; i≤n; i++)
      { if(Jumpcommand(L[i]))
            JT[JumpTarget(L[i])]=1; }   // set store to 1
int j=1;
BB[0] = 1;
for(int i=1; i≤n; i++)
   { if(Jumpcommand(L[i-1]) || JT[i]==1) j++;
     BB[i]=j
   }
   return BB;
```

Exercise 4

The following C program is given:

```
1   main()
2   {
3       int i;
4       int a[10];
5
6       i = 0;
7       while (i < 10)
8       {
9           if (a[i] == 0)
10              a[i] = a[i] + (i * 3);
11          else
12              a[i] = a[i] + ((i * 2) / (a[i] * 3);
13
14          i = i + 1;
15      }
16      ...
17  }
```

(a) Translate this program in 3-Address-Code.

(b) Determine the basic blocks.

(c) Draw the *Degenerated Control-Flow Graph*.

(d) Draw the DAGs of all basic blocks.

a) BB_1 ⟶ $i := 0$ — if x relop y goto L

BB_2 ⟶ if $i >= 10$ goto [15]

BB_3 [$t_1 := a[i]$

if $t_1 == 0$ goto [10]

BB_4 [else | $t_1 := i * 2$

$t_3 := t_1 * 3$

$t_4 := t_2 / t_3$ ($t_5 := t_2 / t_3$)

$t_1 := t_1 + t_4$ ($t_1 := t_1 + t_3$)

goto [12]

BB_5 [line 10 if | $t_2 := i * 3$

$t_1 := t_1 + t_2$

BB_6 [$a[i] = t_1$

$i := i + 1$

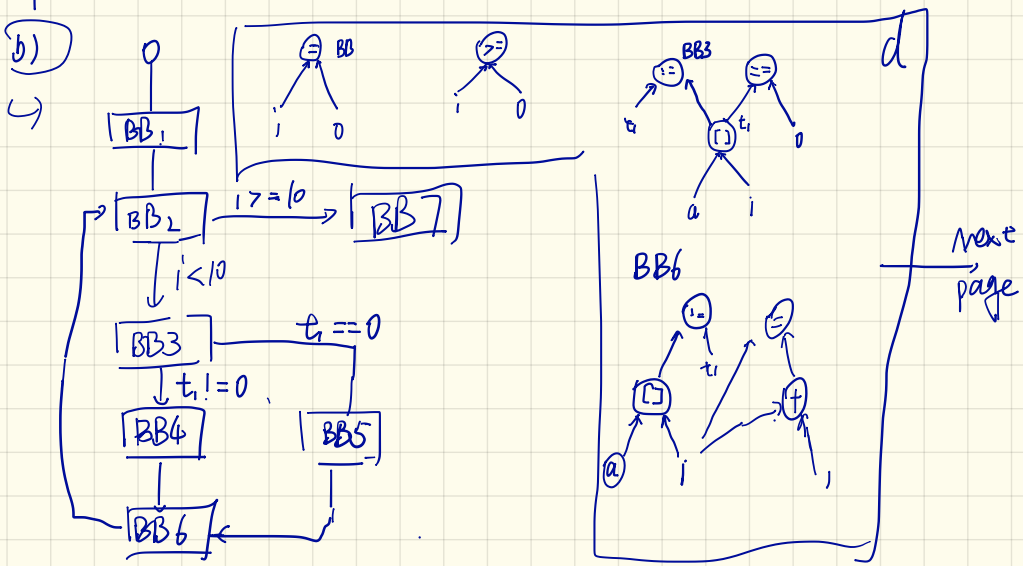
goto [2]

BB_7 ⟶ 15 ...

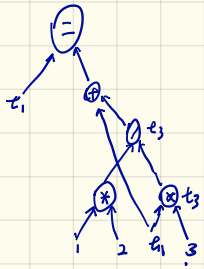
Start of BB_i
1) Start of Program
2) Target of Jump
3) Line after Jump

End of BB_i
1) Jump Line

b)



d)



Department of Computer Engieneering

Hardware/Software Codesign I
Winter Term 2015/16

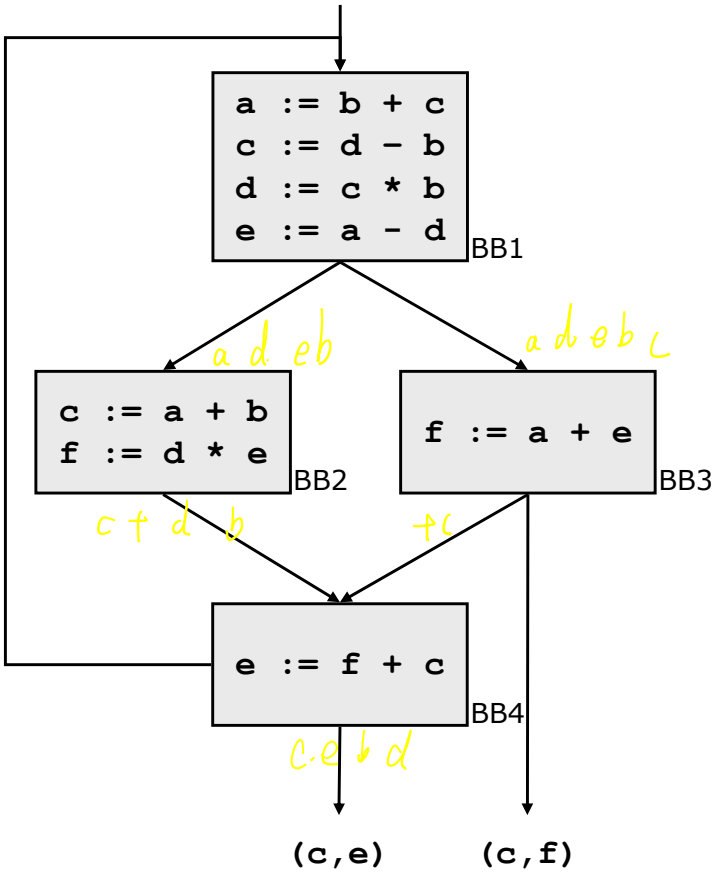
Prof. Dr. Wolfram Hardt, Dipl.-Inf. Michael Nagler

Lesson 6

Register Binding

Exercise 1

The following degenerated control-flow graph describes a program (branch conditions excluded):



(a) Determine the *Life Times* of the variables in *BB1*. Use the algorithm that was presented in the lecture. Assume, that the variables a, d and e are active at the end of *BB1*.

(b) Identify the active variables at each block entrance and each block end.

(c) Determine the life times of all variables during the runtime of the whole loop. Draw the result to a diagram.

(d) Note the costs that can be saved if a register is allocated for a variable during the runtime of the whole loop. Define the equation to calculate the saved costs during one run of the loop.

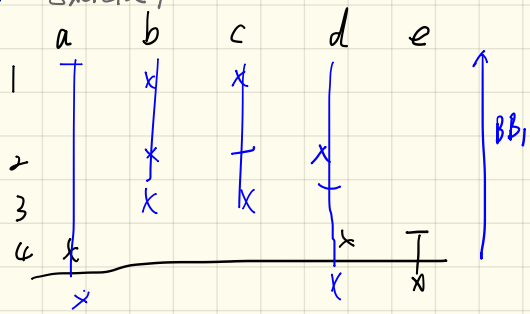
(e) Determine a *register binding*. Use the algorithm of *Usage Counters*. Assume, that 5 registers are available on the target system.

(f) Draw the *Register Conflict Graph* of the given program.

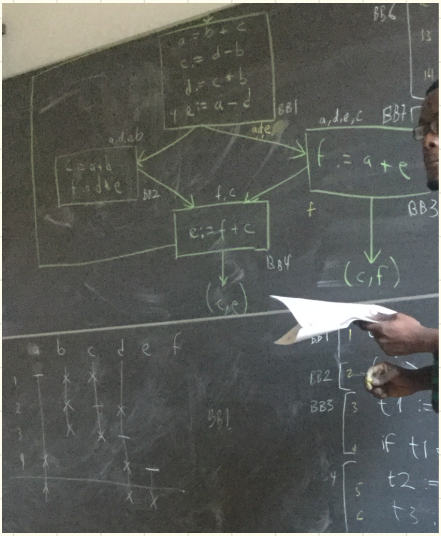
(g) Determine a register binding by using the *Graph Colouring*. Assume, that the target system has 5 registers.

(h) What is a *memory spill*? What variables should be selected for a spill?

Task 6 Exercise 1



X : active

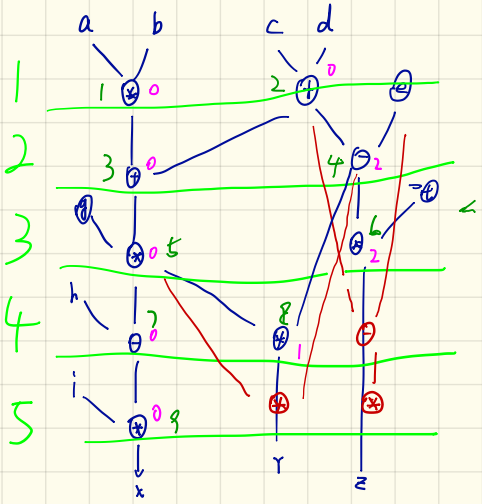


1. - a = to
   - 5. x x = a
   - 6 x y = -a
   - 7 x d = 4 x a
   - x x = a
   - - a = -a
   - x d = 4 x a

- ADD a, R0 (2) → ADD R1, R0 (1) → Save 1 cost
- if a has been defined in BB and is active at the end
  MOV R0, a (2)

Lesson 7

Exercise 1 (a)



(b) ASAP as soon as possible

ALAP as late as possible

(c) ASAP

(d) Mobility = ALAP − ASAP

the others can not move

so mobility = 0

Exercise 2

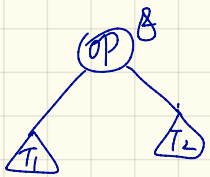
cstep	schedule nodes of	PList[0] = MUL	PList = ADD
0	3 - 8	1	2
1	5 - 8		3 4
2	6 - 8	5	4
3	8	8. 6	7
4	—	8. 6	
5		6	
6			

Lesson 8

Exercise 1

(a)
(1) MOV #1, R_0
(2) MOV y, R_0
(3) MOV x, R_0
ADD #1, R_0
(4) MOV b, R_0
~~MOV c, R_1~~
MUL c, R_1
MOV a, R_1
SUB R_0, R_1
(5) MOV b, R_0
ADD c, R_0
MOV a, R_1
DIV R_0, R_1
MOV e, R_0
ADD f, R_0
MOV d, R_2
SUB R_0, R_2
SUB R_2, R_1

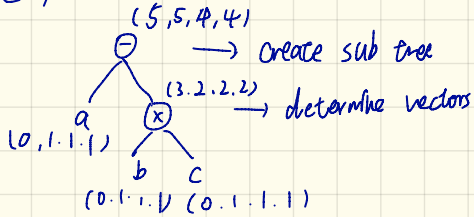
(b)



& = T_1 op T_2

- determine cost vectors
- instruction order
- generate code

(c)



MC R_1 R_2 R_1

if 1Reg to op MUL 1 Reg: $R_i = R_i$ op M_j (1 1 0) = 2

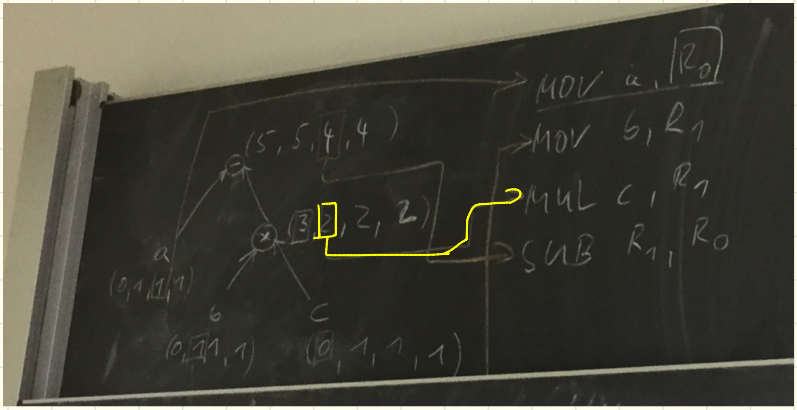
2Reg $R_i = R_i$ op M_j = 2

$R_i = R_i$ op R_j (1 1 1) = 3

3Regs = 2Regs.

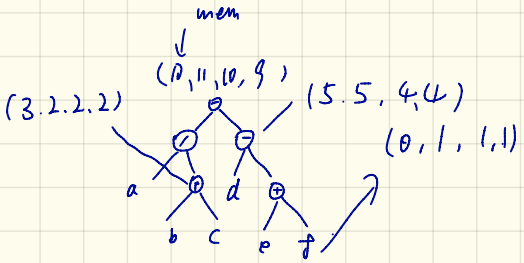
memory = (min(1,2,3 Reg) + 1 MOV = 3

move to memory



黄色线

(5)



1Reg $R_i = R_i \ op \ M_j$

5 1 5 = 11

2Reg ~~=11~~

$R_i = R_i \ op \ R_j$

4 1 5 = 10

$R_i = R_i \ op \ M_j = 10$

3Reg = $R_i = R_i \ op \ M_j = 10$

$R_i = R_i \ op \ R_j = 9$

4 1 4

memory = min(1, 2.5) + 1 = 10

mov a R_0

mov b, R_1 左 右

ADD c, R_1

DIV R_1, R_0

mv d, R_1

mov e, R_2

ADD f, R_2

SUB R_2, R_1

SUB R_1, R_0

Lesson 9

Exercise 1 (b)

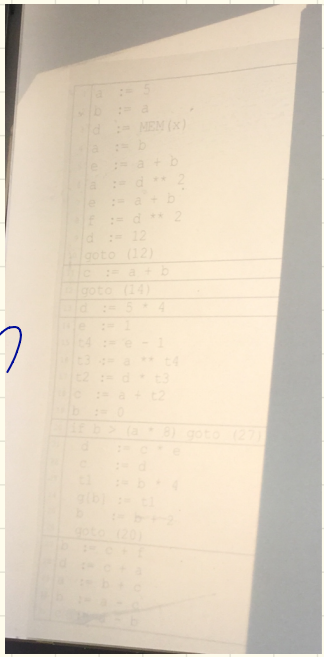
(local optimision) — basic blocks as unit

```
a = b+c
b = a-d
c = b+c
d = a-d  → d = b
```

```
x := y+0 x (2 x 4/y)  → x = y
y = y x 1  → nop (no operation)
x = x+0  → nop
```

```
a = 5   ↓ simplization
b = a x 4  = b = 20
```

(a) →



Globe optimisation

```
x = t1
a(t2) = t3
a(t4) = x   →   a[t4] = t1
```

```
while (i < (x*4+2))        a = (      )
{...}                      while ( i<a)
     ↓ take out a =        {- - -}
```

→

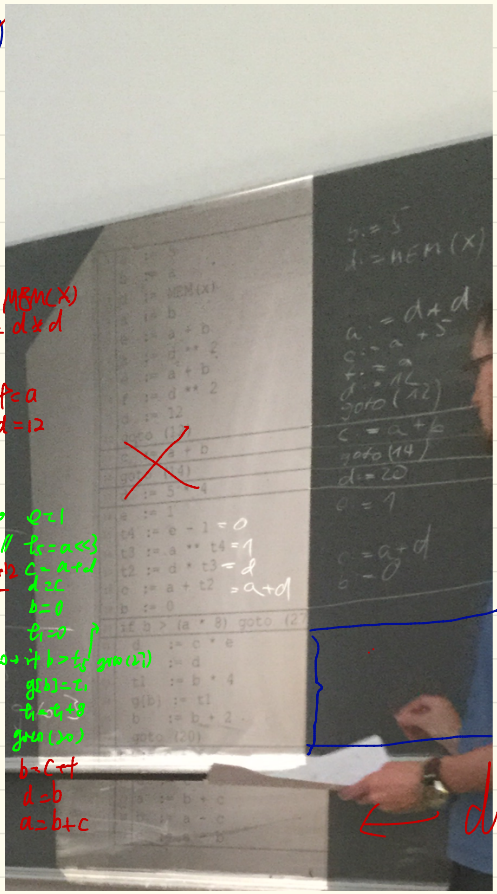
```
j = n
j = j-1
t4 = 4 x j
tj = a(t4)
if t5 > 0 goto 3
```

→

```
j = n
t4 = j x 4
j = j-1
t4 - t4 - 4
t5 = a[t4]
if t5 > 0 goto 3
```

(C)

```
for(i=0; i<100; i=i+1)
{ a[i] = i;
  a[i+1] = i+1
}
```



machine code

mov R, a → nop

mov a, R

MUL R_0, 8 → SHL R_1, 3

SQR R_0, → MUL R_0, R_0

(1) jump 3 → Jump N

(2) ;

(3) jump N