| Name | |
|---|---|

| Matrikel-nummer | |
|---|---|

*Dauer der Prüfung: 120 Minuten.*
*Es sind **alle** Aufgaben mit allen Teilaufgaben zu lösen.*
*Lösen Sie die Aufgaben direkt auf diesen Aufgabenblättern.*

***Gesamtpunktzahl: 30 Punkte.*** ***Viel Erfolg!***

---

**Exercise 1** (10 Points)

a) **True**
b) **False: It depends on the current state as well.**
c) **True.**
d) **True**
e) **False: have longer**
f) **False: it depends**
g) **True: Pipeline performance is related to throughput, not the latency of instructions.**
h) **True**
i) **False: they can occur**
j) **True**

## **Exercise 2** (20 Points)

a) rs=20=$10100_2$, rt=21=$10101_2$, imm=0xC=0000'0000'0000'$1100_2$, opcode=0x23=$100011_2$ ; I-Type : opcode, rs, rt, imm => 100011|10100|10101|0000'0000'0000'1100

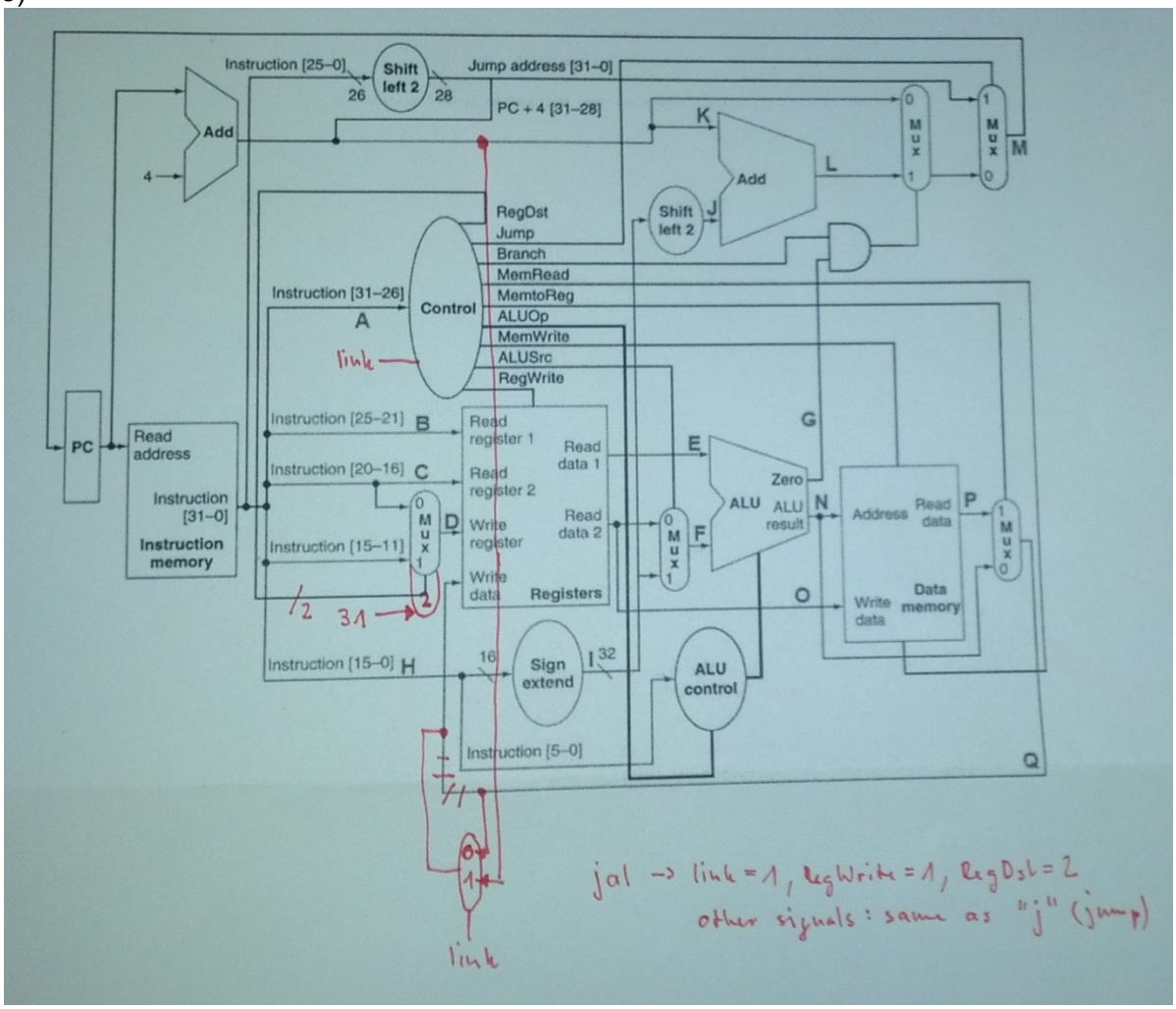
b) effective Memory location to fetch from

c) RegDest=1, Jump=0, Branch=0, MemRead=1, MemtoReg=1, MemWrite=0, ALUSrc=1, RegWrite=1

d)

| **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** |
|---|---|---|---|---|---|---|---|---|---|
| 0X23 | 20 | 21 | 0 | 21 | 0xC | 0xC | 0xC | 0x1004'0000 | 4 |
| **K** | **L** | **M** | **N** | **O** | **P** | **Q** | **R** | **S** | **T** |
| 0xC | 0 | 0x1004'000C | 0x2FFF'0003 | ->N | 0x1008 | 0x30 | 0x1038 | 0 | ->P |

e)




**JAL -- *Jump and link***

| Description: | Jumps to the calculated address and stores the return address in $31 |
|---|---|
| Operation: | $31 = PC + 8 (or nPC + 4); PC = nPC; nPC = (PC & 0xf0000000) \| (target << 2); |
| Syntax: | jal target |
| Encoding: | `0000 11ii iiii iiii iiii iiii iiii iiii` |

**Exercise 3** (10 Punkte)

a)

***Large memories are slow and fast memories are small***
***Try to give the illusion that memory is large, cheap and fast***

b)

***Static Random Access Memory bzw. Dynamic Random Access Memory***
***SRAM: speed (fast)***
***DRAM: size (large, high density), low power, cheap***

c)

i.

***2 GHz x 50 ns = 2 x 10^9 /s x 50 x 10^-9 s = 100***
***1.5 (cycle) + (0.2 (datamemops/instr) x 0.05 (miss/datamemop) x 100 (cycle/miss) + 0.01 (miss/datamemop) x 100 (cycle/miss)) = 1.5 + (1 + 1) = 3.5***

ii.

***2 / 3.5***

iii.

***1.5 + (0.2 x 0.05 x 20 + 0.2 x 0.005 x 100 + 0.01 x 20 + 0.005 x 100) = 1.5 + (0.2 + 0.1 + 0.2 + 0.5) = 2.5***

**Exercise 4** (10 Punkte)

1.

```
Loop:     lw   $t0, 0($s1)     5
          lw   $t1, 200($s1)   5
          addu $t2, $t0, $t1   4
          sw   $t2, 0($s1)     4
          addi $s1, $s1, -4    4
          bne  $s1, Loop       3
                              ----------
                              Total: 25
```

| 2a. Instruction | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lw $t0, 0($s1) | IF | ID | EX | M | W | | | | | | | | | | | | |
| lw $t1, 200($s1) | | IF | ID | EX | M | W | | | | | | | | | | | |
| addu $t2, $t0, $t1 | | | IF | ID | * | Ex | M | W | | | | | | | | | |
| sw $t2, 0($s1) | | | | IF | * | ID | EX | M | W | | | | | | | | |
| addi $s1, $s1, -4 | | | | | | IF | ID | EX | M | W | | | | | | | |
| bne $s1, Loop | | | | | | | IF | ID | EX | M | W | | | | | | |

**1. Data ($t1) for the ADDU is ready after "M" stage of the LW $t1. During the "WB" stage the requested operand will be written to the $t1 and operation register of the ALU.**
**2. ID stage for the SW is delayed because it is busy with ADDU.**

**Number of cycles: 11**

**3b) Reordering:**

```
lw   $t0, 0($s1)
lw   $t1, 200($s1)
addi $s1, $s1, -4
addu $t2, $t0, $t1
bne  $s1, Loop
sw   $t2, 0($s1)
```

**Perfect Pipeline: Number of cycles 10**

**Aufgabe 5** (15 Punkte)

*a)*

```
1          addi $t0, $zero, 0    // offset = 0
2          addi $t1, $zero, 20   // max = 5 * 4
```

```
3           addi $t2, $zero, 4    // index for if = 1 * 4
4 loop:     add $t3, $s0, $t0     // address = m + offset = i * 4
5           addi $t4, $zero, 3    // result = 3 = 0 + 3
6           bne $t0, $t2, GoOn    // if i * 4 = 1 * 4
7           muli $t4, $t4, 2      // result = 2 * result
8 GoOn:     sw $t4, 0($t3)        // m[i] = result
9           addi $t0, $t0, 4      // offset += 4 equals i+=1
10          blt $t0, $t1, loop    // offset < max ?
```

i.

***3 of 6***

ii.

***4 of 6***

b)

***2-bit prediction performs worse, since the predictors are initialized "wrong" and for the first three times*** **predict taken** ***und*** **predict not taken** ***alternate.***

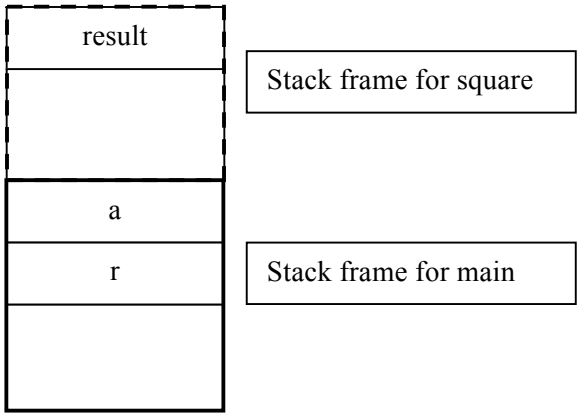
## **Aufgabe 6** (20 points)

a)

8
4
0xABCD
0x5678

b)

i) 2, result variable is allocated on “square” stack frame and is deallocated after return to main -> r is not altered
ii) local variables only:




iii) void square(int x, int* result) { *result = x*x; }
square(a, &r);

c)

1004
‘A’,‘E’, ‘C’, ‘A’
1005, ‘C’, ‘E’
1004, ‘C’, ‘D’, ‘D’, ‘E’

d)

15
1+4*3+2=15
#define AREA(a, b) (a)*(b) => ((1+4)*(3+2)=25)

e)

24 (1*2*3*4 = 24)
26 (5+6+7+8 = 26)

**Aufgabe 7** (20 points)

**Solution:**

**a) 0101-0110=1111**

**b) Risc Processors have a fixed width instruction encoding with fixed position encoding (i.e., all instructions 32 bits wide with register fields in fixed positions). This allowed for simpler control, especially for pipelined and superscalar implementations. CISC Processors use a variable number of bytes to encode instruction. This was needed because of the complex instructions that were supported.**

**c) Register addressing – operand is in a register; Base (displacement) adressing: operand is at the memory location whose address ist he sum of a register and a 16-bit constant contained withing the instruction. Immediate addressing: operand is a 16-bit constant contained withing the instruction. Details See slides V03 page 45.**

**d) j label; |op|26-bit| 4 from PC + 2 zeros shift Slided V03 page 20**

**e) Instruction level parallelism is a measure of the average number of the operations in a program can be performed simultaneously V09 page 5**

**f) Compulsory misses; Conflict misses; Capacity misses. Slides V11 page 26**

**g).i) T-T-T-T-N-T-T-T-N CORRECT:5 WRONG :4**
**ii) T-T-T-T-T-T-T-T-T CORRECT:7 WRONG:2**

**h) To keep track of a call stack.**

**i). Is an anomaly when a program overwrites adjacent memory. Example: inapropriate use of strcpy in C.**

**j)**

**k) Direct mapped: for each item of data at the lower level, there is exactly one location in the cache where it might be- so lots of items at the lower level must share locations in the upper level. Fully associative: data from any address can be stored in any cache location.**