Klausur Mikroprozessortechnik
29. März 2010

Name: ..... Vorname: .....

Matr.-Nr: ..... Studiengang: .....

Hinweise:

- Bitte füllen Sie vor dem Bearbeiten der Aufgaben das Deckblatt sorgfältig aus.
- Zur Klausur zugelassen sind ausschließlich Schreibutensilien, aber **kein Taschenrechner und kein eigenes Papier**!
- Schreiben Sie bitte auf alle Zusatzblätter Ihren Namen und Ihre Matrikelnummer und beginnen Sie jede Aufgabe auf einem neuen Blatt.
- Betrugsversuche führen zum **sofortigen** Ausschluss der Klausur.
- Lösungen in Bleistift können nicht gewertet werden.
- Voraussetzung für die volle Punktzahl ist immer, dass der Lösungsweg vollständig erkennbar ist.
- Die Bearbeitungszeit beträgt **120** Minuten.

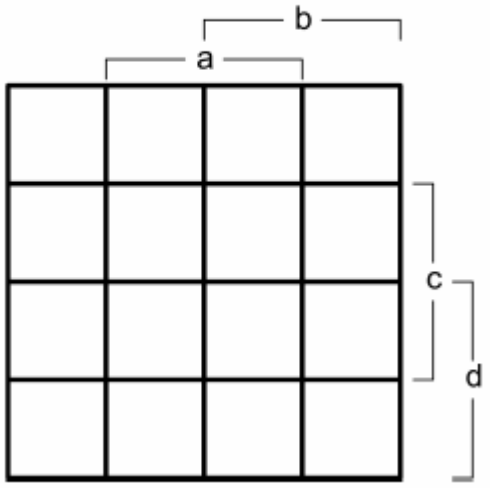
AUFGABE	PUNKTE
1	
2	
3	
4	
Gesamtpunkte	
Note	

Aufgabe 1)

a) Schaltnetzentwurf , CMOS (3 Punkte)

a) Boolesche Funktionen
Füllen Sie die Wahrheitstabelle für folgende logische Verknüpfung aus und geben Sie die minimale disjunktive Normalform an. **(2 Punkte)**

$$\overline{c} \cdot \overline{(a+b)(b \rightarrow a)} + \overline{\overline{a}cc}$$



Hinweis:
Für die Implikation gilt:

a	b	$(b \rightarrow a)$
0	0	1
0	1	0
1	0	1
1	1	1

b) Zeichnen Sie die entsprechende C-MOS Schaltung für folgende Funktion **(1 Punkt)**
$y = \overline{a}dc + b\overline{c}$

b) Cache (3 Punkte)

c) Was ist der Unterschied zwischen einem „direct-mapped“ und einem vollassoziativen Cache? Begründen Sie kurz und erläutern Sie ganz allgemein, wie bei einem Zugriff die Adressen der einzelnen Cache-Zeilen in der jeweiligen Implementierung bestimmt werden. **(2 Punkte)**

d) Gegeben Sei ein 2-fach satzassoziativer Cache mit 2 Sätzen. Ermitteln Sie die Anzahl der Fehlzugriffe für Speicherzugriffe mit der gegebenen Blockreihenfolge:

1,7,1,5,7

wenn der Cache am Anfang leer ist und eine LRU-Ersetzungsstrategie verwendet wird. Zeigen Sie bitte den Lösungsweg.
(1 Punkt)

Aufgabe 2)
Schaltwerksentwurf (7 Punkte)

In der folgenden Aufgabe soll ein Zahlenschloss als synchrones Schaltwerk entworfen werden. An dem Schaltwerk werden dazu synchron zum Takt Binärzahlen zwischen 0 und 3 als Eingabevektor x=(x1,x0) angelegt. Das Schloss soll genau dann öffnen, wenn die zuletzt angelegten Ziffern der Folge „031“ entsprachen (zuerst ein 0, dann eine 3 und zuletzt eine 1).

a) Entwerfen Sie ein geeignetes Zustandsdiagram! **(1 Punkt)**

b) Bestimmen Sie die minimalen Übergangsfunktionen! **(2 Punkte)**

c) Bestimmen Sie die Beschaltungsfunktionen (muss nicht minimal sein) für die Flip-Flops. Das Schaltwerk soll ausschließlich mit SR-Flip-Flops realisiert werden. **(2 Punkte)**

d) Realisieren Sie ein JK-Flip-Flop mithilfe eines D-Flip-Flops und zeichnen Sie die Schaltung! Wie könnten Sie damit auf einfachste Weise ein T-Flip-Flop realisieren? **(2 Punkte)**

Aufgabe 3)
Datenpfad (9 Punkte)

Für die folgenden Aufgaben stehen Ihnen die entsprechenden Datenpfade der MIPS-Mehrzyklenimplementierung im Anhang zur Verfügung. Lassen Sie sich bei irreparablen Fehlversuchen beim Bearbeiten des Datenpfads neue Abbildungen geben!

a) Beschreiben Sie die Funktion der Steuersignale IorD und PCWriteCond. **(1 Punkt)**

b) Welche Adressierungsart wird bei unbedingten Sprüngen verwendet? Beschreiben Sie, wie sich die Zieladresse bei diesen Sprüngen zusammensetzt. **(1 Punkt)**

c) Der Datenpfad soll nun durch zwei neue Befehle um zusätzliche Funktionalität erweitert werden.

gca $reg1
swap $reg1,$reg2

Befehlsbeschreibung:

Mit dem „gca“ (get current address) Befehl soll die aktuelle Befehlsadresse des aufrufenden Befehls in dem übergebenen Register gespeichert werden. Die neue „swap“ Instruktion hingegen soll die Inhalte der beiden Register $reg1 und $reg2 vertauschen.

Implementierung:
Erweitern Sie, sofern nötig, den Datenpfad im Anhang um zusätzliche Elemente und zeichnen Sie diese in die Abbildung ein, damit die Befehle korrekt ausführbar sind.

Hinweis: Alles was funktioniert ist richtig, es gibt durchaus mehrere Möglichkeiten!

(1 Punkt)

c) Ausführung:

Erläutern Sie kurz, was in welcher Ausführungsstufe passiert und setzen Sie alle Steuerleitungen. Nutzen Sie die dafür vorgegebenen Tabellen.

Überlegen Sie sich ein geeignetes Befehlsformat.
Denken Sie daran: Sie haben alle Freiheiten, solange der Befehl richtig ausgeführt wird. **(4 Punkte)**

gca $reg1

Stufe	Aktionen und Steurleitungen
1	
2	
3	
4	
5	

PC Write Cond	PC Write	IorD	Mem Read	Mem Write	Mem ToReg	IR Write	PC Source	ALU Op	ALU SrcA	ALU SrcB	Reg Write	Reg Dst		

swap $reg1,$reg2

Stufe	Aktionen und Steurleitungen
1	
2	
3	
4	
5	

PC Write Cond	PC Write	IorD	Mem Read	Mem Write	Mem ToReg	IR Write	PC Source	ALU Op	ALU SrcA	ALU SrcB	Reg Write	Reg Dst		

b) Befehlssatzerweiterung durch Pseudobefehle

Im Folgenden soll der Befehlssatz des MIPS-Assemblers durch die Pseudobefehle **„not“** sowie **„blt“** erweitert werden. Gehen Sie davon aus, dass der Datenpfad nur die Befehle

and, or, add, sub, slt, nor , beq

beherrscht.

Es gilt:

not	not $s1,$s2	$s1 = not ($s2)
blt	blt $s1,$s2,25	if ($s1<$s2) GoTo PC+4+100

Setzen Sie die 2 Pseudobefehle um.

Hinweis:
Das Register $at (Assembler Temporary) wird für temporäre Werte bei der Umsetzung von Pseudobefehlen vom Assembler genutzt. Greifen Sie, falls nötig, auf dieses Register zurück.

(2 Punkte)

Aufgabe 4)
MIPS Assembler (8 Punkte)

a) Betrachten Sie für den ersten Teil dieser Aufgabe das folgende Codesegment. Beschreiben Sie mit einer mathematisch/logischen Formel die Funktionalität des gegebenen Programms. Kommentieren Sie alle relevanten Programmzeilen.
(3 Punkte)

```
1: main:        li      $t0,0x30
2:              xor     $s0,$s0,$s0
3: loop:        beq     $t0,$zero,exit
4:              add     $t1,$t0,$t0
5:              srl     $t2,$t0,1
6:              add     $t1,$t1,$t2
7:              addi    $t1,$t1,-1
8:              sll     $t2,$t1,2
9:              add     $t2,$t2,$t1
10:             add     $s0,$s0,$t2
11:             addi    $t0,$t0,-1
12:             j       loop
13: exit:       sw      $s0,0($sp)
```

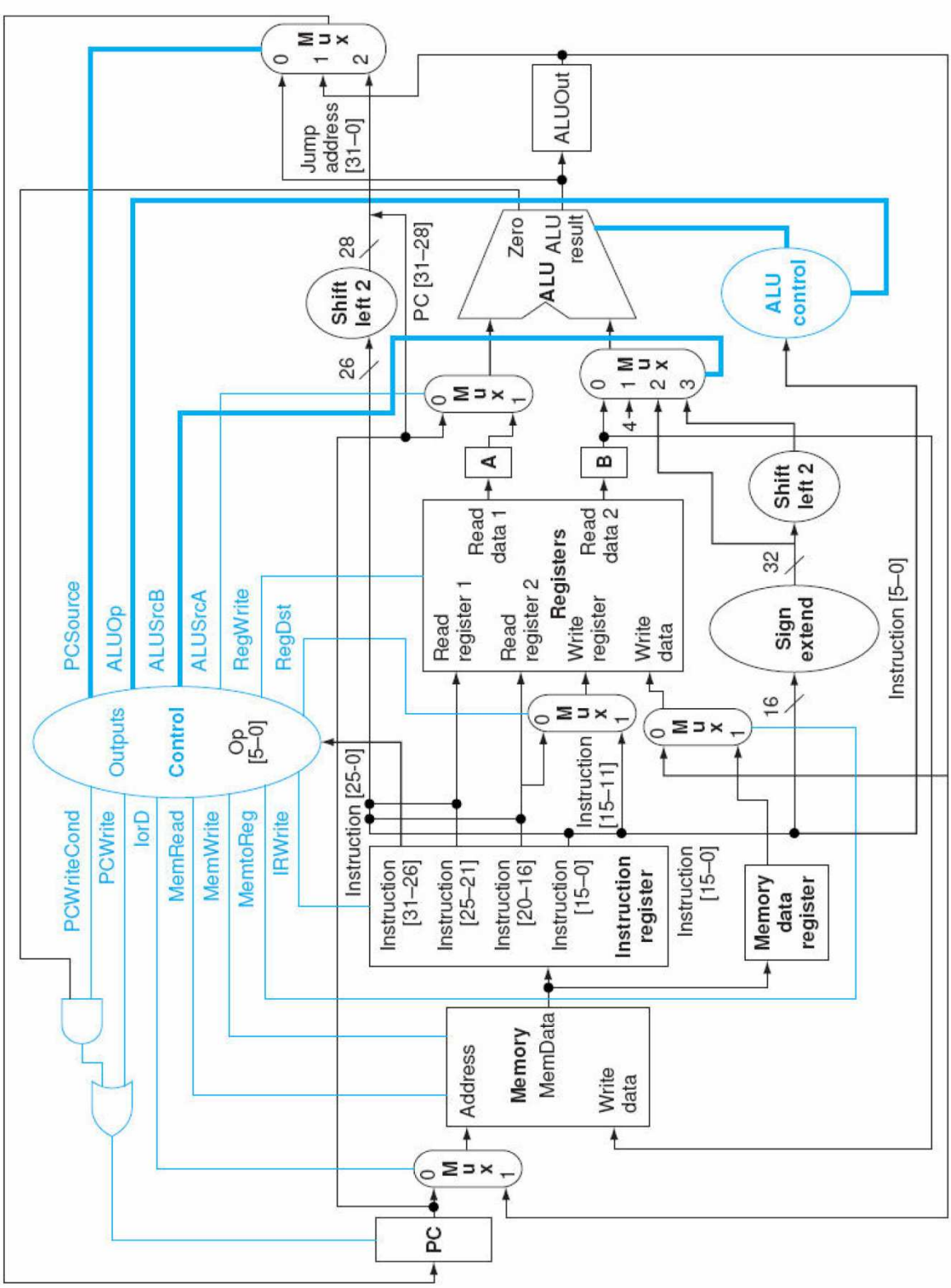
b) Gegeben sei ein 32-Bit Wort-Array im Datenspeicher. Implementieren Sie eine Funktion in MIPS Assembler, die als Parameter die Basisadresse und die Länge des Arrays bekommt. Die Funktion soll den Maximalwert des Arrays und die Anzahl der Maxima zurückgeben.

Implementieren Sie nur den Funktionsrumpf. Benutzen Sie dabei ausschließlich $s-Register! Beachten Sie dabei die MIPS-Konventionen für Unterprogramme.

Kommentieren Sie jede Programmzeile!
(5 Punkte)

Anhang)

Mehrzyklendatenpfad



MIPS Registerübericht und Wertetabelle der ALUSteuerungseinheit.

Name	Register Number	Usage	Preserved on call
$zero	0	the constant value 0	n.a.
$at	1	reserved for the assembler	n.a.
$v0-$v1	2-3	value for results and expressions	no
$a0-$a3	4-7	arguments (procedures/functions)	yes
$t0-$t7	8-15	temporaries	no
$s0-$s7	16-23	saved	yes
$t8-$t9	24-25	more temporaries	no
$k0-$k1	26-27	reserved for the operating system	n.a.
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return address	yes

Opcode	ALUOp	Operation	funct-field	ALU Operation	ALU ControlInput
LW	00	load word	******	Addition	0010
SW	00	store word	******	Addition	0010
Branch on equal	01	branch on equal	******	Subtraction	0110
R-Command	10	add	100000	Addition	0010
R-Command	10	subtract	100010	Subtraction	0110
R-Command	10	and	100100	And	0000
R-Command	10	or	100101	Or	0001
R-Command	10	set on less than	101010	less than	0111

Auszug MIPS Befehlsreferenz

add	add $s1,$s2,$s3	$s1 = $s2 + $s3
subtract	sub $s1,$s2,$s3	$s1 = $s2 - $s3
add immediate	addi $s1,$s2,100	$s1 = $s2 + 100
add unsigned	addu $s1,$s2,$s3	$s1 = $s2 + $s3
subtract unsigned	subu $s1,$s2,100	$s1 = $s2 - $s3
add immediate unsigned	addiu $s1,$s2,100	$s1 = $s2 + 100
move from coprocessor register	mfc0 $s1, $epc	$s1 = $epc
multiply	mult $s2, $s3	Hi, Lo = $s2 x $s3
multiply unsigned	multu $s2, $s3	Hi, Lo = $s2 x $s3
divide	div $s2, $s3	Lo = $s2 : $s3, Hi = $s2 % $s3
divide unsigned	divu $s2, $s3	Lo = $s2 : $s3, Hi = $s2 % $s3
move from Hi	mfhi $s1	$s1 = Hi
move from Lo	mflo $s1	$s1 = Lo
load word	lw $s1, 100($s2)	$s1 = Memory[$s2 + 100]
store word	sw $s1, 100($s2)	Memory[$s2 + 100] = $s1
load half unsigned	lhu $s1, 100($s2)	$s1 = Memory[$s2 + 100]
store half	sh $s1, 100($s2)	Memory[$s2 +100] = $s1
load byte unsigned	lbu $s1,100($s2)	$s1 = Memory[$s2 + 100]
store byte	sb $s1,100($s2)	Memory[$s2 + 100] = $s1
load upper immediate	lui $s1, 100	$s1 = 100 * 2^16
and	and $s1,$s2,$s3	$s1 = $s2 & $s3
or	or $s1,$s2,$s3	$s1 = $s2 | $s3
nor	nor $s1,$s2,$s3	$s1 = ~($s2 | $s3)
and immediate	andi $s1,$s2,100	$s1 = $s2 & 100
or immediate	ori $s1,$s2,100	$s1 = $s2 | 100
shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10
shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10
branch if equal	beq $s1,$s2,25	if ($s1==$s2) GoTo PC+4+100
branch if not equal	bne$s1,$s2,25	if($s1!= $s2) GoTo PC+4+100
branch if greater than	bgt $s1,$s2,25	if($s1> $s2) GoTo PC+4+100
banch if greater than or equal	bge $s1,$s2,25	if($s1>= $s2) GoTo PC+4+100
banch if less than	blt $s1,$s2,25	if($s1<s2) GoTo PC+4+100
banch if less than or equal	ble $s1,$s2,25	if($s1< =$s2) GoTo PC+4+100
set on less than	slt $s1,$s2,$s3	if($s2<$s3) $s1=1 else $s1=0
set less than immediate	slti $s1,$s2,100	if($s2<100) $s1=1 else $s1=0
set less than unsigned	sltu $s1,$s2,$s3	if($s2<$s3) $s1=1 else $s1=0
set less than immediate unsigned	sltiu $s1,$s2,100	if($s2<100) $s1=1 else $s1=0
jump	J 2500	GoTo 10000
jump register	jr $ra	GoTo $ra
jump and link	jal 2500	$ra = PC + 4, GoTo 10000

Flip Flop Übergangs- und Beschaltungsfunktionen

	D – FF	SR – FF	JK – FF	T - FF
Übergangsfunktion	$u^+ = d$	$u^+ = \bar{u}s + u\bar{r}$	$u^+ = j\bar{u} + \bar{k}u$	$u^+ = \bar{t}u + t\bar{u}$
Beschaltungsfunktion	$d = u^+$	$s = u^+\|_{u=0}$ $r = \overline{u^+}\|_{u=1}$	$j = u^+\|_{u=0}$ $k = \overline{u^+}\|_{u=1}$	$t = u^+ \oplus u$