Meilensteine des SpartanMC 18

Ereignis	Datum	Bearbeiter
1. Gedanken zu Spartan	14.02.2005	Schöne, Hochberger
Befehlssatz und Kodierung	08.03.2005	Schöne
Start eines Assembler als großer Beleg	19.04.2005	Gebhardt
Start für die Standart Peripherie als großer Beleg	07.06.2005	Kunze
Web Ordner /SpartanMC/ angelegt.	16.07.2005	Schöne
Assembler 1. Version	15.08.2005	Gebhardt
Assembler 2. Version	03.09.2005	Gebhardt
Implementierung des Befehlssatzes auf xc3s400	05.09.2005	Haßler, Hochberger
Assembler Quellen mit Namen ***.a18**	18.09.2005	Schöne
Texte1.a18 → HALLO auf 7 Segment Anzeige	18.09.2005	Schöne
SpartanMC HEX Format *.sph mit **sr2spMC**	23.09.2005	Schöne
Operandenreihefolge vertauschen zur Einsparung von Hardware.	27.09.2005	Haßler, Gebhardt
SpartanMC_ASS mit vertauschten Operanden	28.09.2005	Gebhardt
Freigabe des Interrupt und RFE Befehl	01.10.2005	Haßler, Schöne
Fehler → RegBase ein Bit zu klein	05.10.2005	Schöne
Test der ansprechbaren 520 Register	11.10.2005	Schöne
und er bewegt sich doch ... → **Video**	11.10.2005	Schöne
Fehler in den Vergleichbefehlen beseitigt. Beim Vergleich von Registern wurde RS1 zerstört.	12.10.2005	Haßler, Schöne
Fehler in den Speicherbefehlen gefunden.	13.10.2005	Schöne
Fehler in den Speicherbefehlen **beseitigt**.	27.10.2005	Haßler, Schöne
Start C Compiler	01.11.2005	Jakschik
Neue Version des Tools sr2spMC	01.11.2005	Schöne
Fehler im movs2i.	03.11.2005	Schöne
Fehler im movs2i beseitigt.	04.11.2005	Schöne
Kein MUX notwendig bei mehr als 1 BLOCK-RAM	09.11.2005	Schöne
MUL Befehl Implementiert	11.11.2005	Schöne
Spartan hat jetzt **2K** Worte **Speicher**	13.11.2005	Schöne
Peripherie Interface	13.11.2005	Haßler
Die UART geht mit 56700 Baud	05.12.2005	Kunze
Monitor mit den Kommandos HELP, Display 9 Bit und Display 18 Bit arbeitet.	12.01.2006	Schöne
Monitor mit Kommando Modify erweitert.	13.01.2006	Schöne
SpartanMC_ASS mit neuen movs2i und movi2s.	16.01.2006	Gebhardt
Laden von Programmen > 1K Worte möglich.	19.01.2006	Schöne
Monitor mit Goto und Load.	24.01.2006	Schöne
Start für den Simulator	03.02.2006	Spickereit
ISE 7.1 unbrauchbar! → 46 Fehler	04.02.2006	Schöne
Fehler in ISE 7.1 → **webpack 8.1** verwenden	07.02.2006	Schöne
SpartanMC mit 14K Mem und 1K DMA Puffer.	10.02.2006	Schöne
JTAG Interface unzuverlässig?	12.02.2006	Schöne
Kein fehlerfreies Laden möglich!	12.02.2006	Schöne

Meilensteine des SpartanMC 18

Ereignis	Datum	Bearbeiter
Neue Implementierung des SpartanMC mit 9 Bit Speicher Blöcken und neuem E/A Interface.	28.02.2006	Haßler
Monitor mit 16 TRAP Einsprüngen.	01.03.2006	Schöne
USB-Teil arbeitet	05.03.2006	Haßler
Make und sr2spMC zum laden des DMA Speichers geändert.	07.03.2006	Schöne
Monitor mit USB-DMA Initialisierung	09.03.2006	Schöne
Interrupt von USB und UART 2 bis 4 getestet.	12.03.2006	Schöne
Änderung UART Statusregister	14.03.2006	Schöne
Paper	23.03.2006	Hochberger
Beleg Kunze 1. Fassung	03.04.2006	Kunze
1. Version SpartanMC_sim von Stefan Spickereit.	25.04.2006	Spickereit
Start der Arbeiten für den CAN Kontroller.	15.05.2006	Hempel
1. Testversion des SpartanSimulator.jar verfügbar.	15.05.2006	Spickereit
SpartanMC_ASS mit movi2s, movs2i, sbits und cbits.	19.05.2006	Gebhardt
Monitor jetzt im Speicher verschieblich.	19.05.2006	Schöne
Die Abstürze bei der Interruptarbeit beseitigt.	31.05.2006	Schöne
Der Befehl RFE ist jetzt implementiert.	31.05.2006	Schöne
IFADDUI und IFSUBUI ohne Vorzeichenerweiterung der Konstante.	31.05.2006	Schöne
MUL jetzt fehlerfrei, aber Fehler im VSIM!	31.05.2006	Schöne
TRAP auf 8 Bit Adresse erweitert.	31.05.2006	Schöne
WIKI Seiten zum SpartanMC angelegt.	01.06.2006	Schöne
UART jetzt mit alle BAUD Raten.	15.06.2006	Schöne
UART mit 8 FIFO Registern arbeitet jetzt auch.	16.06.2006	Schöne
RESET beim Einschalten und bei PROG.	23.06.2006	Schöne
RESET jetzt intern erzeugt.	30.06.2006	Schöne
SpartanMC_ASS Version 2.4.1	06.07.2006	Gebhardt
SpartanSimulator jetzt mit UART	10.07.2006	Spickereit
SpartanSimulator jetzt auch mit Interruptkontroller	20.07.2006	Spickereit
SpartanMC_ASS mit Makros.	03.08.2006	Gebhardt
SPI+I2C slave funktionieren.	17.08.2006	Kunze
Start für jConfig den neuen Konfigurations- Tool	17.08.2006	Haßler
Beleg zur Standard E/A abgeschlossen.	30.08.2006	Kunze
Fehler im UNSIGNED Vergleich.	24.09.2006	Schöne
Fehler im UNSIGNED Vergleich behoben.	26.09.2006	Schöne
CAN Kontroller abgeschlossen	14.10.2006	Hempel
Weiter Bearbeitung von jConfig durch D. Lippmann	19.10.2006	Lippmann
Monitor mit neuer Speicheraufteilung für den C-Compiler	19.10.2006	Schöne
SpartanMC 18 jetzt mit 1024 Registern.	20.10.2006	Schöne
Shift mit CC.	25.10.2006	Schöne
SUB und ADD jetzt mit OV im CC.	26.10.2006	Schöne
Neue **Webseite** zum SpartanMC 18 **als Frame**.	28.10.2006	Schöne
SpartanSimulator mit allen Hardware Änderungen.	02.11.2006	Schöne
Neue Assembler Version.	03.11.2006	Gebhardt
Neuer Monitor mit 36 Bit E/A Rufen als TRAP.	04.11.2006	Schöne

Meilensteine des SpartanMC 18

Ereignis	Datum	Bearbeiter
Neue WEB Frame zum Prozessor, zur Peripherie und zu den Tools.	05.11.2006	Schöne
add36 und sub36 als Unterprogramm.	14.11.2006	Schöne
Mulu18 und div3618 als Unterprogramm.	15.11.2006	Schöne
Fehler in initram.tcl gefunden.	03.12.2006	Schöne
Fehler beseitigt und Laden von mehr als 2K eingebaut.	04.12.2006	Schöne
Auch in initdmaram.tcl waren die gleichen Fehler.	05.12.2006	Schöne
Einstellung des PROM im Konfig Werkzeug notwendig.	06.12.2006	Schöne
Belegverteidigung Steffen Kunze	06.12.2006	Kunze
Compiler 0.2 und neue Unterordner im Verzeichnis SpartanMC auf Bert.	18.12.2006	Jakschik
1. Testversion von jConfig	20.12.2006	Lippmann
Beleg von Herrn Gebhardt als 1. Entwurf	07.01.2007	Gebhardt
Dokumentation zum C- Compiler im WIKI	09.01.2007	Jakschik
monitor_print("Hallo");	10.01.2007	Jakschik
Neue Version des Assembler	10.01.2007	Gebhardt
Neue Version von sr2spMC	10.01.2007	Schöne
Neue Version von spmon.a18	10.01.2007	Schöne
Neue Version von jConfig	15.01.2007	Lippmann
Monitor-Dienste sind jetzt in C verfügbar.	16.01.2007	Jakschik
WIKI Seiten zu den E/A Modulen	17.01.2007	Kunze
WIKI Seiten zum Assembler	18.01.2007	Gebhardt
Neue Version des Assembler	19.01.2007	Gebhardt
ROTL () Makro in Funktion.inc	21.01.2007	Schöne
Neue Version des Assembler	21.01.2007	Gebhardt
SpartanMC jetzt **mit jConfig** erstellen.	14.02.2007	Schöne
Aufwand an LUT mit jConfig ermittelt.	08.03.2007	Schöne
i^2c Slave ADS1000 erfolgreich angesprochen.	14.03.2007	Schöne
JTAG-Modul	19.03.2007	Meyer
JTAG-Modul mit geändertem Interrupt	23.03.2007	Meyer
i^2c Slave 24LC24 mit Modify Programm angesprochen.	23.03.2007	Schöne
TPA Modul	25.03.2007	Meyer
JTAG mit Korrektur-Shifts	30.03.2007	Meyer
SpartanMC mit 8K Speicher	07.05.2007	Schöne
Neuer Monitor mit Laden des Speichers auch oberhalb des Monitors.	20.05.2007	Schöne
SPI für jConfig im SVN.	09.06.2007	Schöne
Alle Unterprogramme als Macro.	18.06.2007	Schöne
DMA Geräte jetzt wie alle I/O Geräte mit einigen zusätzlichen Signalen eingebunden.	24.06.2007	Schöne
jConfig mit Expertmode	24.06.2007	Lippmann
jConfig für **neue DMA Anbindung** geändert.	05.07.2007	Lippmann
Alle Spartanquellen für neue DMA Anbindung im SVN	23.10.2007	Schöne
Neuer Button im jConfig	24.10.2007	Lippmann
Alle Interface von Herrn Kunze im jConfig	09.11.2007	Schöne
SPI Testprogramm für Antje	14.11.2007	Schöne

Meilensteine des SpartanMC 18

Ereignis	Datum	Bearbeiter
Ise9.2 auf Piggy und Tier	15.11.2007	Hochberger
Interface für **Rotations Taster**	29.11.2007	Schöne
di_peri nur 18 Bit für IRQ, UART, Timer und i2c	03.12.2007	Schöne
Veränderungen am SPI für DAC LTC2624, LTC6912-1 ...	05.12.2007	Schöne
Neue Version jConfig	11.12.2007	Lippmann
Beschleunigung von Funktionen -- Diplomarbeit	21.12.2007	Schindhelm
Neue Version jConfig	08.01.2008	Lippmann
USB 2.0 UTMI Baustein usb3250	11.01.2008	Hochberger
Start der Arbeiten zum USB 2.0	16.01.2008	Mikava
Clock.v geändert	06.02.2008	Lippmann
Neues Dateiformat ***.spho**, neuer Monitor und sr2spMC ...	11.02.2008	Schöne
Wirksamkeit der User Constraints	15.02.2008	Lippmann
Verbindungen SpartanMC EVB_USB3250	17.02.2008	Mikava
LCD am SpartanMC	23.02.2008	Schöne
Leiterplatine für USB3250 – **Abbruch der Arbeiten**	24.02.2008	Mikava
LCD und ROT_TA mit Interrupt getestet.	19.03.2008	Schöne
Neuer **FIFO** für die UART und andere Interface	19.03.2008	Lippmann
Unsicherheit im SBITS INT in der controll.v beseitigt	20.03.2008	Schöne
Neue testbench.fdo für VSIM 6.3f war notwendig	28.03.2008	Pöckel
Verzögerung in RESET für VSIM in spartanmc.v eingebaut.	29.03.2008	Schöne
Neue testbench.udo mit Speicheranzeige für 3 Blöcke.	30.03.2008	Schöne
Neues jConfig ohne Fehler in der system.v	01.04.2008	Lippmann
Neue testbench.v für einen SPI-Slave	03.04.2008	Schöne
Neue testbench.v für einen SPI-Master und Slave	04.04.2008	Schöne
UART Rx Fehlerfrei noch mit shiftstatus Port	31.05.2008	Schöne
C-Compiler mit Schalter „-target=spartan/moni“	16.06.2008	Schöne
CAN Interface für jconfig aufbereitet.	27.06.2008	Schöne
C: Vor dem Block mit "compiler.inc" muss ein .even stehen!	11.07.2008	Schöne
C: Neues MAKE für LCC18 und ASS18	12.07.2008	Engelmann
C: Negieren eines Wertes um einen Befehl verkürzt. Vorzeichen lose Tests mit einer Konstanten gehen so nicht --> Entfernt.	17.07.2008	Schöne
Jetzt haben alle Interface und CAN nur 1 OR-Gate Port	21.07.2008	Schöne
UART FIFO Synchronisation jetzt fehlerfrei	24.07.2008	Schöne

SpartanMC 18

Inhalte:

Der SpartanMC 18 – Prozessor

- Befehlssatz
- Adressierung
- Programmierung

1. Versuch 17.02.2005

SpartanMC 18

Der SpartanMC 18 – Prozessor

- Lade/Speicher-Architektur
- 16(+16) * 18 Bit Register r0 bis r15 (r0‘ bis r15‘) und ein CC Register
- Tauschregister für r0 bis r7 und für r8 bis r15 getrennt umschaltbar
- Befehlssatz mit 2 Adressen
- Adressierung des Speichers mit 4 Bit Displacement
- Vergleichs (SET) Befehle schreiben das Ergebnis in das CC Register
- Sprünge mit 12 Bit Offset für unbedingte und CC Befehle
- Sprünge mit 8 Bit Offset bei Testung eines Registers

SpartanMC 18

Hauptoperationscodes des SpartanMC

$IR_{17..15}$ \ $IR_{14..12}$	000	001	010	011	100	101	110	111
000	SPEZIAL1	SPEZIAL2	J	JAL	BEQZ	BNEZ	BEQZC	BNEZC
001	ADDI	ADDUI	SUBI	SUBUI	ANDI	ORI	XORI	MULI
010	RFE	TRAP	JR	JALR	SLLI		SRLI	SRAI
011	SEQI	SNEI	SLTI	SGTI	SLEI	SGEI	IFADDUI	EXLR
100	LB	LH	L9	L18	LBU	LHU		MOVI
101	SB	SH	S9	S18	LHI0	LHI4	LHI8	LHI12
110	SEQUI	SNEUI	SLTUI	SGTUI	SLEUI	SGEUI	IFSUBUI	EXHR

Erweiterte Operationscodes func im R-Befehlsformat des SpartanMC (SPEZIAL1 und SPEZIAL2)

IR_3 \ $IR_{2..0}$	000	001	010	011	100	101	110	111
0	ORCC	ANDCC	MOVI2C	MOVC2I	SLL	MOV	SRL	SRA
1	SEQU	SNEU	SLTU	SGTU	SLEU	SGEU		
0	ADD	ADDU	SUB	SUBU	AND	OR	XOR	MUL
1	SEQ	SNE	SLT	SGT	SLE	SGE	MOVI2S	MOVS2I

SpartanMC 18

Die Befehle entsprechen im wesentlichen den DLX Befehlen, sie haben aber nur 2 Operanden.
Neue Befehle des SpartanMC sind:

BEQZC	**name**	**Springe wenn der Inhalt des CC-Register Null ist.**
BNEZC	**name**	**Springe wenn der Inhalt des CC-Register nicht Null ist.**
MULI	**Rd, i**	**18 Bit Multiplikation mit einem 8 Bit Immediate**
MUL	**Rd, Rs**	**18 Bit Multiplikation**
MOVI	**Rd, i**	**Rd mit i Vorzeichenerweitert laden.**
MOV	**Rd, Rs**	**Rd mit Inhalt von Rs laden.**
LHI0	**rd, i**	**für r0 - r3 **
LHI4	**rd, i**	**für r4 - r7 |__ Laden von 10 Bit Immediate in die oberen 10 Bit eines Register**
LHI8	**rd, i**	**für r8 - r11 |**
LHI12	**rd, i**	**für r12 - r15 /**
ORCC	**Rs**	**CC <-- CC or Rs**
ANDCC	**Rs**	**CC <-- CC and Rs**
MOVI2C	**Rs**	**CC <-- Rs**
MOVC2I	**Rd**	**Rd <-- CC**
EXLR		**Austausch von r0 bis r7 mit den Schattenregistern r0' bis r7'**
EXHR		**Austausch von r8 bis r15 mit den Schattenregistern r8' bis r15'**

SpartanMC 18

Befehle zur Verarbeitung von Daten im 9(8) Bit Format:

```
L9          Rd, disp(Rs)        ;Lädt die unteren 9 Bit, wenn CC = 0 ist und sonst die oberen 9 Bit
                                ;Nach der Ausführung wird CC in das Gegenteil geändert.
                                ;In Rd stehen die 9 Bit mit „0“ erweitert
S9          disp(Rs1), Rs2      ;speichert in unteren 9 Bit, wenn CC = 0 ist und sonst in die oberen 9 Bit
                                ;Nach der Ausführung wird CC in das Gegenteil geändert.

LB und LBU                      ;Die Bytebefehle arbeiten wie die Befehle L9 und S9 aber beim laden
SB                              ;kann zwischen „0“ und „VZ“ Erweiterung gewählt werden.

IFADDUI     Rd, Imm             ;IF  CC != 0  → Rd = Rd + Imm
IFSUBUI     Rd, Imm             ;IF  CC != 0  → Rd = Rd – Imm
```

SpartanMC 18

Die 5 Spartan-Befehlsformate sollen an Hand der folgenden Befehle in die 18-**Bit-Befehlswörter** codiert werden.

- ALU-Befehle

```
add     r2,r1
andi    r4,0xff
seq     r5,r6          ;Setze CC-Register
lhi     r0,0x3ff       ;10 Bit nach High laden für   r0 ... r3
lhi     r5,0x255       ;10 Bit nach High laden für   r4 ... r7
lhi     r8,0x111       ;10 Bit nach High laden für   r8 ... r11
lhi     r14,0xff       ;10 Bit nach High laden für r12 ... r15
```

- Verzweigebefehle

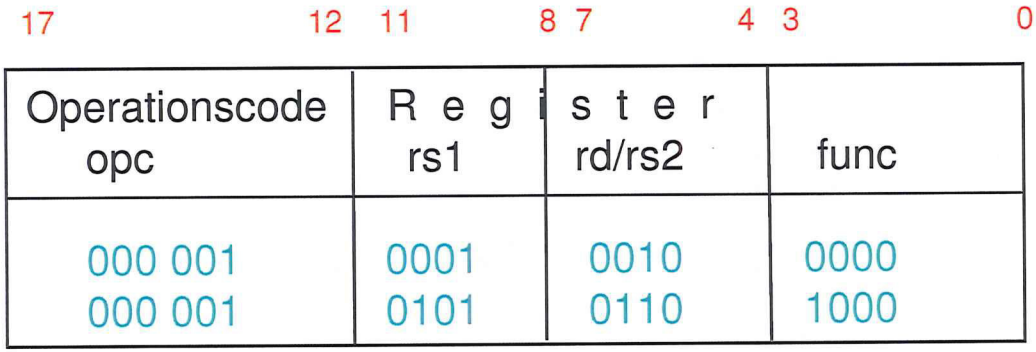
```
bnez    r3,loop        ;  8 Bit Offset
beqzc   m0001          ;12 Bit Offset,  testet CC-Register
j       addr1
```

- Lade/Speicher-Befehle

```
lb      r5,0x2(r4)
```

SpartanMC 18

R-Type (Register)



IR

add

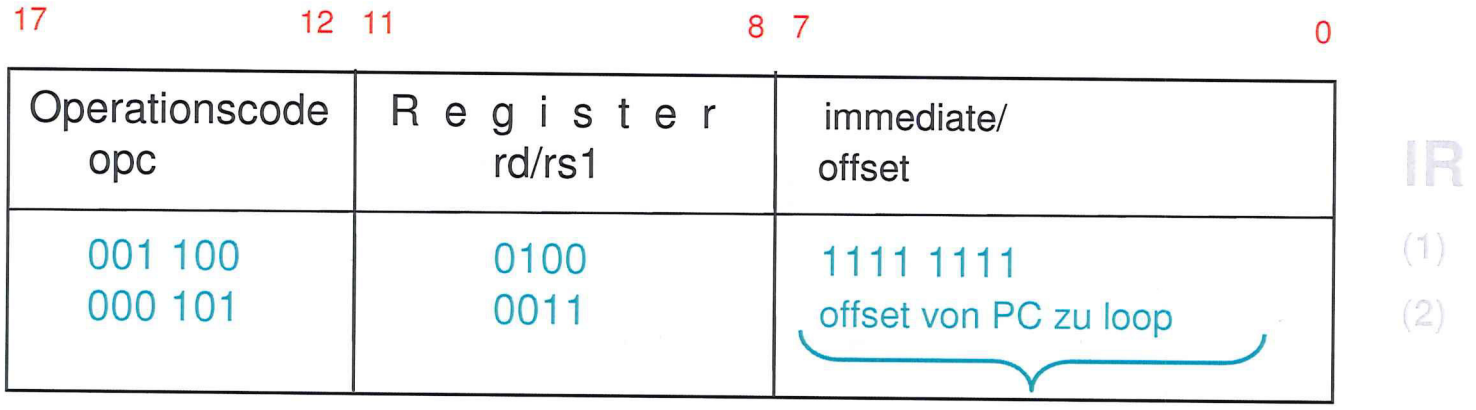
seq

ALU-Befehl

```
add     r2,r1     ; r2 <---- r2 + r1
seq     r5,r6     ; cc <---- r5 == r6
```

SpartanMC 18

I-Type (Immediate)



(1) ALU-Befehl `andi r4,0xff` ; r4 ⟵ r4 & 0x000ff

(2) Verzweigebefehl `bnez r3,loop` ; PC ⟵ PC + (IR_7^{10} ## $IR_{7...0}$)
; offset von PC+1 zu loop (mit VZ)

SpartanMC 18

L-Type (Load Immediate High)

17 … 14	13 … 10	9 … 0
Operationscode opc	R e g i s t e r rd	immediate
001 1	0000	11 1111 1111
001 1	0101	10 0101 0101
001 1	1000	01 0001 0001
001 1	1110	00 1111 1111

IR

(1)
(2)
(3)
(4)

(1) ALU-Befehle
(2)
(3)
(4)

```
lhi     r0,0x3ff    ; r0  <---  0x3ff00
lhi     r5,0x255    ; r5  <---  0x25500
lhi     r8,0x111    ; r8  <---  0x11100
lhi     r14,0xff    ; r14 <---  0x0ff00
```

SpartanMC 18

M-Type (Memory)

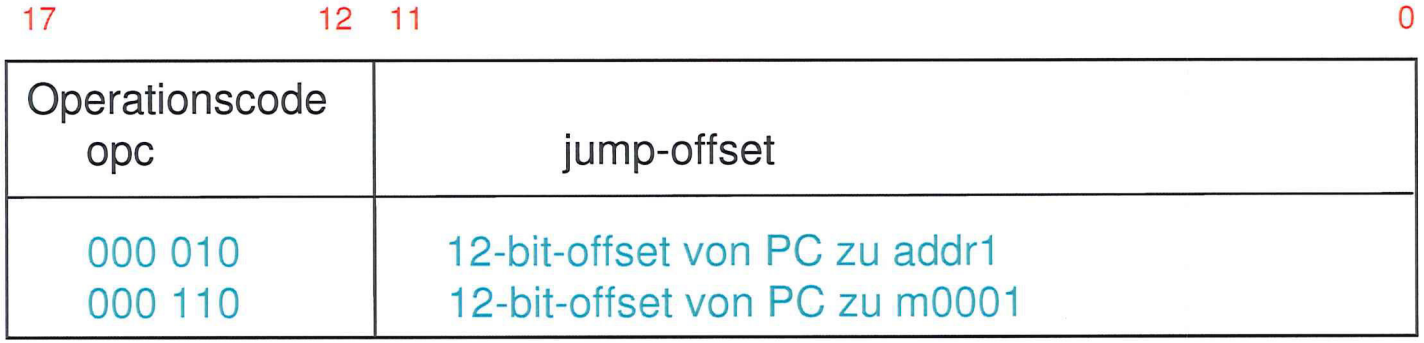
17 – Operationscode opc	12 11 – Register rs1	8 7 – Register rd/rs2	4 3 – 0 displacement
100 000	0100	0101	0010

IR

Lade/Speicher-Befehl lb r5,0x2(r4) ; r5 ⟵ $M[r4 + 0x2]_7^{10}$ ## M[r4 +0x2]

SpartanMC 18

J-Type (Jump)



(1) Verzweigebefehl `j addr1` ; PC ⟵ PC + (IR_{11}^{6} ## $IR_{11...0}$)

(2) `beqzc m0001` ; PC ⟵ PC + (IR_{11}^{6} ## $IR_{11...0}$), wenn cc = 0

; offset von PC+1 zu m0001 (mit VZ)

SpartanMC 18

Für die folgende Aufgabe soll nun eine Befehlsfolge ermittelt werden.

a = b + c;
d = e – f; a,b,c,d,e,f sind 18-Bit Worte im Speicher mit den Labels A,B,C,D,E,F

Der nachstehend angegebenen Rahmen soll dafür angewendet werden.

```
        .text       0x100
start:  lhi         r1,0X102        ;
        ...........                 ;  Laden Adresse A nach R1
        ...........                 ;  Laden b nach R2
        ...........

        trap        0               ; steht hier für eine HALT-Anweisung, liefert  Ausschrift "trap 0"

        .data       0x10200
A:      .w18        0x.....
B:      .w18        0x.....
C:      .w18        0x.....
D:      .w18        0x.....
E:      .w18        0x.....
F:      .w18        0x.....
```

SpartanMC 18

```
        .text       0x100
start:  lhi         r1,0X102    ;
        ori         r1,0x10     ;   Laden Adresse A nach R1
        l18         r2,1(r1)    ;  Laden b nach R2
        l18         r3,2(r1)
        add         r3,r2
        s18         0(r1),r3    ;  a = b + c
        l18         r2,4(r1)
        l18         r3,5(r1)
        sub         r2,r3
        s18         3(r1),r2
        trap        0           ; steht hier für eine HALT-Anweisung, liefert  Ausschrift "trap 0"

        .data       0x10210
A:      .w18        0x.....
B:      .w18        0x.....
C:      .w18        0x.....
D:      .w18        0x.....
E:      .w18        0x.....
F:      .w18        0x.....
```

SpartanMC 18

Zwei gleich große Datenfelder mit 18-Bit-Daten der Länge n sollen auf Gleichheit untersucht werden.

```
                .text
start:


                .data
                .........
adr_n:          .byte       ; Länge der Felder F1 und F2
ergebnis:       .byte       ; Ergebnis: 0 für ≠ , 1 für =

adr_F1:         .w18        ; Adresse von Feld1
                ..........
adr_F2:         .w18        ; Adresse von Feld2
```

SpartanMC 18

```
; Test auf Gleichheit von 2 gleich großen Datenfeldern (mit 18-Bit Daten)
            .text
start:      lhi     r5,adr_F1>>8        ; die oberen 10 Bit von F1 in r5 laden.
            ori     r5,adr_F1&0xff      ; Adr. von F1 in r5
            lhi     r6,adr_F2>>8
            ori     r6,adr_F2&0xff      ; Adr. von F2 in r6
            lhi     r7,adr_n>>8
            ori     r7,adr_n&0xff
            lb      r7,0(r7)            ; Länge der Felder in r7, Zähler für Test
            lhi     r8,ergebnis>>8
            ori     r8,ergebnis&0xff
loop:       l18     r9,0(r5)
            l18     r10,0(r6)
            seq     r9,r10              ; cc = 1 wenn gleich, sonst cc = 0
            beqzc   store               ; unterschiedliche Worte, Vergleichen beenden
            addi    r5,0x1
            addi    r6,0x1
            subi    r7,1
            bnez    r7,loop
store:      movc2i  r10
            sb      0(r8),r10
            trap    0
```

SpartanMC 18

```
            .data                   ; Datenbereich
            .........
adr_n:      .byte       n           ; Länge der Felder F1 und F1
ergebnis:   .byte       0xxx        ; Ergebnis: 0 für ≠ , 1 für =

adr_F1:     .w18        f1          ; Adresse von Feld1
            ..........
adr_F2:     .w18        f2          ; Adresse von Feld2
```

SpartanMC 18

```
; Berechnung von Y=X!, Y soll im Speicher an Adresse adr_y
; liegen, R5 enthält bereits X (es gelte X>1)

            .text
begin:      movi    r5,6                ; Y = 6!
            mov     R6,R5
            SUBI    R6,1
loop:       MULT    R5,R6
            SUBI    R6,1
            BNEZ    R6,loop
            LHI     R7,adr_y>>8
            ORI     R7,adr_y&0xFF
            S18     0(R7),R5
            TRAP    0

            .data
adr_y:      .w18    0
```

SpartanMC 18

; Ausgabe einer Zeichenkette ab dem Symbol „text1“ bis zu einer Null

```
            .text
begin:      LHI     R7,text1>>8
            ORI     R7,text1&0xFF
            MOVI    R0,1
            MOVI2C  R0              ;CC := 1
loop:       L9      R1,0(R7)        ;Zeichen laden, toggle CC
            BEQZ    R1,ende_tx
            IFADDUI R7, 1           ;R7 := R7 + 1 wenn CC != 0
            JAL     chout           ;Zeichen auf Konsole anzeigen
            J       loop
ende_tx:    TRAP    0

            .data
text1:      .asciiz „Hallo World\n“ ;Zeichen liegen im 9 Bit Abstand. Wird von
                                    ;text1 mit L18 geladen, steht im Rd 0x09061
```

SpartanMC 18

; 8 Bit Summe über eine Byte Liste bilden, Ende bei 0

```
            .text
begin:      LHI     R7,liste1>>8
            ORI     R7,liste1&0xFF
            MOVI    R0,1
            MOVI2C  R0              ;CC := 1
            XOR     R2, R2          ;Summe löschen
loop:       LBU     R1,0(R7)        ;Byte laden, toggle CC
            BEQZ    R1,ende_su
            IFADDUI R7, 1           ;R7 := R7 + 1 wenn CC != 0
            ADD     R2,R1           ;Summe bilden
            MOVC2I  R1              ;CC in R1 retten
            SGTI    R2,255          ;CC = 1 wenn größer 255
            IFADDUI R2,1            ;Übertrag addieren
            ANDI    R2,255          ;Nur 8 Bit übrig lassen
            MOVI2C  R1              ;CC wieder laden
            J       loop
ende_su:    TRAP    0               ;Summe in R2

            .data
liste1:     .byte   1,2,3,4,5,6     ;Byte liegen im 9 Bit Abstand. Wird mit L18 von
            .byte   7,8,9,0         ;liste1 geladen steht im Rd 0x0202
```

SpartanMC 18

Inhalte:

Der SpartanMC 18 – Prozessor

- Befehlssatz 2. Versuch 15. 6. 2005
- Adressierung
- Programmierung

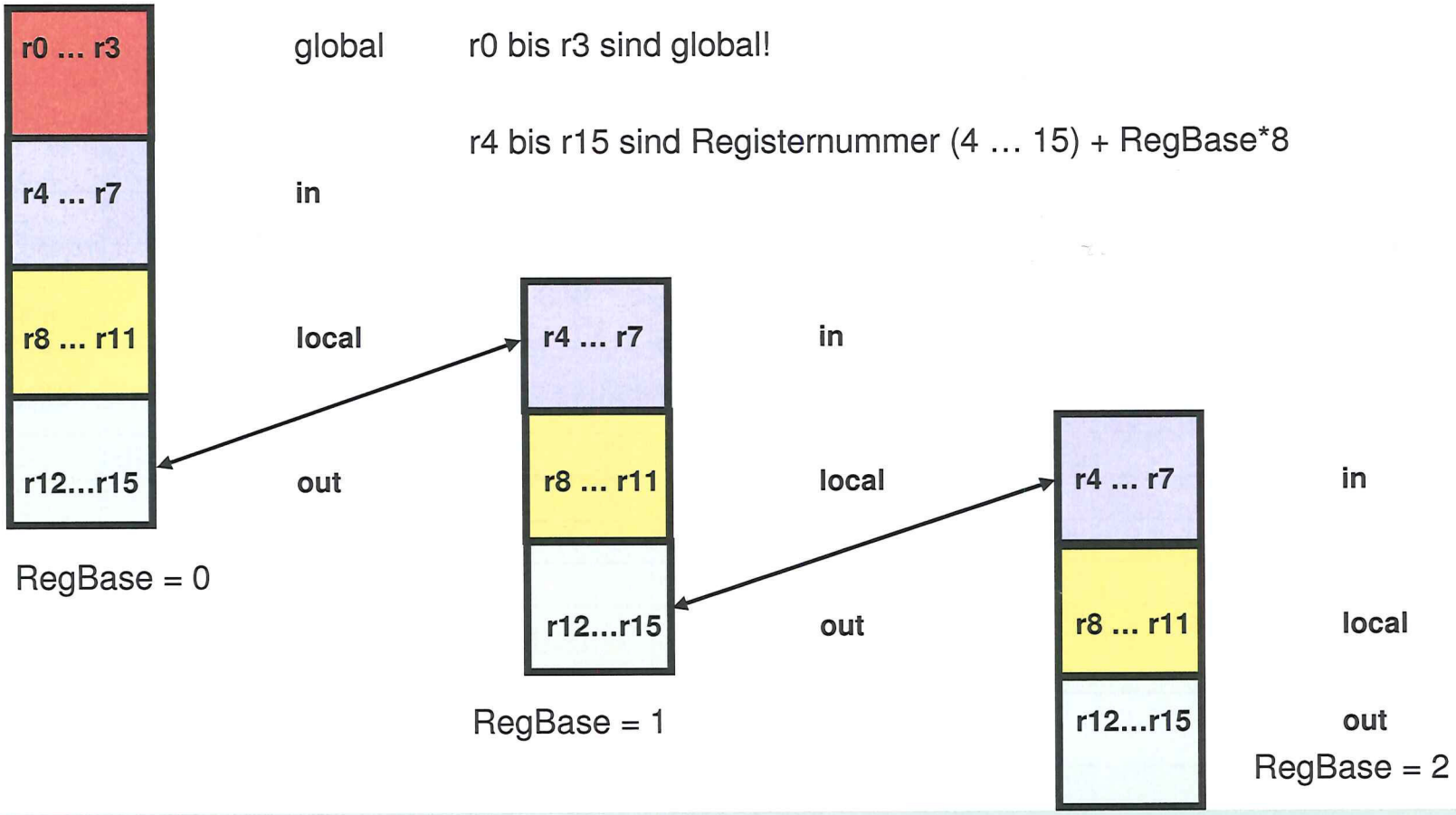
SpartanMC 18

Der SpartanMC 18 – Prozessor

- Lade/Speicher-Architektur
- 16(1016) * 18 Bit Register r0 bis r15. r0 bis r3 sind global. R4 bis r15 werden bei UP Aufrufen mit einem Offset von jeweils 8 im Fenster verschoben.
- und ein CC, MM, EI und RegBase Register. (RegBase kann nicht im Blockram stehen)
- Befehlssatz mit 2 Adressen
- Adressierung des Speichers mit 5 Bit Displacement
- Vergleichs (SET) Befehle schreiben das Ergebnis in das CC Register
- Sprünge mit 13 Bit Offset für unbedingte und CC Befehle
- Sprünge mit 9 Bit Offset bei Testung eines Registers
- Konstanten mit 9 Bit

SpartanMC 18

Register Fenster des SpartanMC 18:



SpartanMC 18

Hauptoperationscodes des SpartanMC

$IR_{17..16}$ \ $IR_{15..13}$	000	001	010	011	100	101	110	111
00	SPEZIAL1	SPEZIAL2	J	JALS	BEQZ	BNEZ	BEQZC	BNEZC
01	ADDI	MOVI	LHI	SIGEX	ANDI	ORI	XORI	MULI
10	L9	S9	L18	S18	SLLI		SRLI	SRAI
11	SEQI	SNEI	SLTI	SGTI	SLEI	SGEI	IFADDUI	IFSUBUI

Erweiterte Operationscodes func im R-Befehlsformat des SpartanMC (SPEZIAL1 und SPEZIAL2)

$IR_{4...3}$ \ $IR_{2..0}$	000	001	010	011	100	101	110	111
00	ORCC	ANDCC	MOVI2C	MOVC2I	SLL	MOV	SRL	SRA
01	SEQU	SNEU	SLTU	SGTU	SLEU	SGEU		
10	CREG	LREG	SREG					
11	SMM	CMM	SCC	CCC	EI	DI		
00	RFE	TRAP	JR	JALR	JRS	JALRS		
01								
10	ADD	ADDU	SUB	SUBU	AND	OR	XOR	MUL
11	SEQ	SNE	SLT	SGT	SLE	SGE	MOVI2S	MOVS2I

SpartanMC 18

Die Befehle entsprechen im wesentlichen den DLX Befehlen, sie haben aber nur 2 Operanden.
Neue Befehle des SpartanMC sind:

```
BEQZC    name     ;Springe wenn der Inhalt des CC-Register Null ist.
BNEZC    name     ;Springe wenn der Inhalt des CC-Register nicht Null ist.

SCC               ;CC <-- 1
CCC               ;CC <-- 0 (ist RESET Belegung)

ORCC     Rs       ;CC  <--  CC or   Rs
ANDCC    Rs       ;CC  <--  CC and  Rs
MOVI2C   Rs       ;CC  <--  Rs
MOVC2I   Rd       ;Rd  <--  CC

MULI     Rd, i    ;18 Bit Multiplikation mit einem 9 Bit Immediate (oberen 18 Bit vom Resultat in Spez. Reg.)
MUL      Rd, Rs   ;18 Bit Multiplikation (oberen 18 Bit vom Resultat in Spez. Reg.)

MOVI     Rd, i    ;Rd mit i ohne Vorzeichenerweiterung laden.
MOV      Rd, Rs   ;Rd mit Inhalt von Rs laden.

CREG              ;Löschen RegBase (ist RESET Belegung)
LREG     Rs       ;Laden RegBase
SREG     Rd       ;Speichern RegBase
```

SpartanMC 18

Befehle zur Verarbeitung von Daten im 9(8) Bit Format:

```
L9      Rd, disp(Rs)      ;Lädt die unteren 9 Bit für ungerade Adressen und bei geraden die oberen
                          ;9 Bit. In Rd stehen die 9 Bit mit „0“ erweitert
S9      disp(Rs1), Rs2    ;speichert in die unteren 9 Bit für ungerade Adressen und bei geraden in
                          ;die oberen 9 Bit.
```

(Die Befehle arbeiten also im Big Endian. Zur Adressierung der oberen 128K 18 Bit Worte muss mm = 1 gesetzt werden.)

```
SMM                       ;mm <-- 1
CMM                       ;mm <-- 0 (ist RESET Belegung)

SIGEX   Rd, Imm           ;Vorzeichen Erweiterung ab Bitnummer in Imm. (Typisch sind 7, 8 und 15)

IFADDUI Rd, Imm           ;IF (CC != 0)  then     Rd <-- Rd + Imm
IFSUBUI Rd, Imm           ;IF (CC != 0)  then     Rd <-- Rd – Imm

EI                        ;EI <-- 1      Interrupt Freigabe
DI                        ;EI <-- 0      Interrupt Sperren (ist RESET Belegung)
```

SpartanMC 18

Unterprogrammbefehle mit Sichern der Register (r11 ist letztes lokales Register zum Retten des PC)

```
JALRS   r4      ;PC <-- R4,                 R11 <-- PC + 1,          RegBase <-- RegBase + 1
JALS    label   ;PC <-- label ,             R11 <-- PC + 1,          RegBase <-- RegBase + 1
JRS     r11     ;RegBase <-- RegBase – 1,                PC <-- R11
```

Unterprogrammbefehl ohne Sichern der Register (r11 ist letztes lokales Register zum Retten des PC)

```
JALR    r5      ;PC <-- R5,                 R11 <-- PC + 1,
JR      r11     ;PC <-- R11
```

SpartanMC 18

Die 4 Spartan-Befehlsformate sollen an Hand der folgenden Befehle in die 18-Bit-Befehlswörter codiert werden.

- ALU-Befehle

```
add     r2,r1
andi    r4,0xff
seq     r5,r6          ;Setze CC-Register
lhi     r14,0xff       ;  9 Bit nach High von r14 laden
```

- Verzweigebefehle

```
bnez    r3,loop        ;  9 Bit Offset
beqzc   m0001          ;13 Bit Offset,  testet CC-Register
j       addr1          ;13 Bit Offset
```

- Lade/Speicher-Befehle

```
l9      r5,0x2(r4)
```

SpartanMC 18

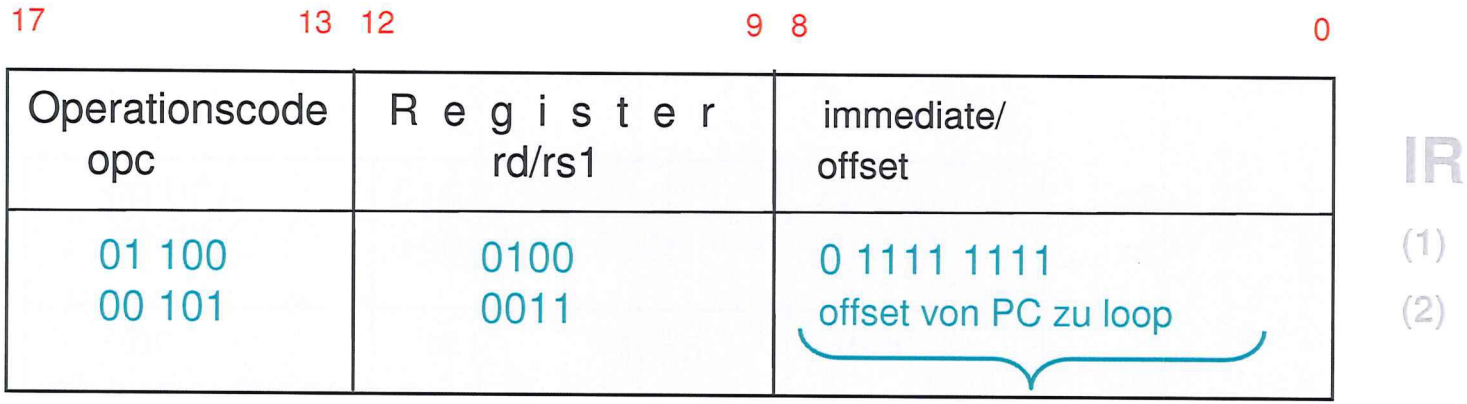
R-Type (Register)

ALU-Befehl

```
add     r2,r1     ; r2 <---- r2 + r1
seq     r5,r6     ; cc <---- r5 == r6
```

SpartanMC 18

I-Type (Immediate)

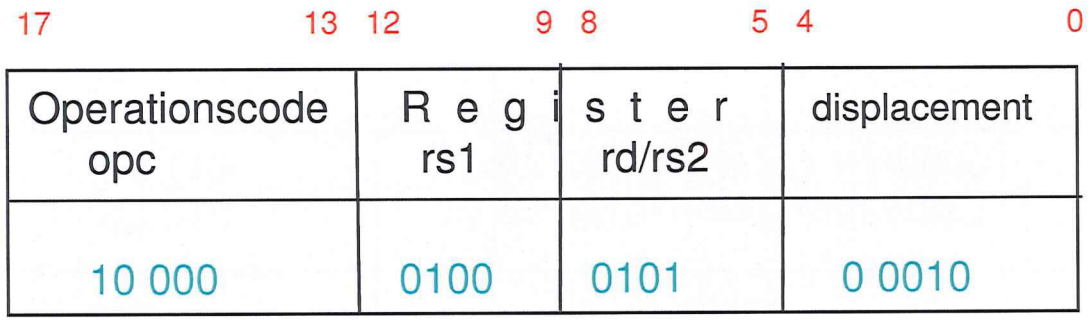


(1) ALU-Befehl `andi r4,0xff` ; r4 ⟵ r4 & 0x000ff

(2) Verzweigebefehl `bnez r3,loop` ; PC ⟵ PC + (IR_8^9 ## $IR_{8...0}$)
; offset von PC+1 zu loop (mit VZ)

SpartanMC 18

M-Type (Memory)



IR

Lade/Speicher-Befehl l9 r5,0x2(r4) ; r5 ⟵ 0^9 ## M[r4 +0x2]

SpartanMC 18

J-Type (Jump)

17 — Operationscode opc	13 12 — jump-offset	0 — IR
00 010	13-bit-offset von PC zu addr1	(1)
00 110	13-bit-offset von PC zu m0001	(2)

(1) Verzweigebefehl `j addr1` ; PC ⟵ PC + (IR_{12}^{5} ## $IR_{12...0}$)

(2) `beqzc m0001` ; PC ⟵ PC + (IR_{12}^{5} ## $IR_{12...0}$), wenn cc = 0

; offset von PC+1 zu m0001 (mit VZ)

SpartanMC 18

Für die folgende Aufgabe soll nun eine Befehlsfolge ermittelt werden.

a = b + c;
d = e – f; a,b,c,d,e,f sind 18-Bit Worte im Speicher mit den Labels A,B,C,D,E,F

Der nachstehend angegebenen Rahmen soll dafür angewendet werden.

```
        .text       0x100
start:  lhi         r1,0X102        ;
                                    ;   Laden Adresse A nach R1
        ...........
        ...........                 ;   Laden b nach R2
        ...........

        trap        0               ; steht hier für eine HALT-Anweisung, liefert  Ausschrift "trap 0"

        .data       0x10200
A:      .w18        0x.....
B:      .w18        0x.....
C:      .w18        0x.....
D:      .w18        0x.....
E:      .w18        0x.....
F:      .w18        0x.....
```

SpartanMC 18

```
        .text       0x100
start:  lhi         r1,0x81     ;
        ori         r1,0x10     ;   Laden Adresse A = 0x10210 nach r1
        l18         r2,1(r1)    ;  Laden b nach r2
        l18         r3,2(r1)    ;  Laden c nach r3
        add         r3,r2
        s18         0(r1),r3    ;  a = b + c
        l18         r2,4(r1)
        l18         r3,5(r1)
        sub         r2,r3
        s18         3(r1),r2
        trap        0           ; steht hier für eine HALT-Anweisung, liefert  Ausschrift “trap 0“

        .data       0x10210
A:      .w18        0x.....
B:      .w18        0x.....
C:      .w18        0x.....
D:      .w18        0x.....
E:      .w18        0x.....
F:      .w18        0x.....
```

SpartanMC 18

Zwei gleich große Datenfelder mit 18-Bit-Daten der Länge n sollen auf Gleichheit untersucht werden.

```
                .text
start:


                .data
                .........
adr_n:          .byte           ; Länge der Felder F1 und F2
ergebnis:       .byte           ; Ergebnis: 0 für ≠ , 1 für =

adr_F1:         .w18            ; Adresse von Feld1
                ..........
adr_F2:         .w18            ; Adresse von Feld2
```

SpartanMC 18

```
; Test auf Gleichheit von 2 gleich großen Datenfeldern (mit 18-Bit Daten)
            .text
start:      lhi     r5,adr_F1>>9        ; die oberen 10 Bit von F1 in r5 laden.
            ori     r5,adr_F1&0x1ff     ; Adr. von F1 in r5
            lhi     r6,adr_F2>>9
            ori     r6,adr_F2&0x1ff     ; Adr. von F2 in r6
            lhi     r7,adr_n>>9
            ori     r7,adr_n&0x1ff
            l9      r7,0(r7)            ; Länge der Felder in r7, Zähler für Test
            lhi     r8,ergebnis>>9
            ori     r8,ergebnis&0x1ff
loop:       l18     r9,0(r5)
            l18     r10,0(r6)
            seq     r9,r10              ; cc = 1 wenn gleich, sonst cc = 0
            beqzc   store               ; unterschiedliche Worte, Vergleichen beenden
            addi    r5,0x1
            addi    r6,0x1
            addi    r7,-1
            bnez    r7,loop
store:      movc2i  r10
            s9      0(r8),r10
            trap    0
```

SpartanMC 18

```
            .data               ; Datenbereich
            .........
adr_n:      .byte     n         ; Länge der Felder F1 und F1
ergebnis:   .byte     0xxx      ; Ergebnis: 0 für ≠ , 1 für =

adr_F1:     .w18      f1        ; Adresse von Feld1
            ..........
adr_F2:     .w18      f2        ; Adresse von Feld2
```

SpartanMC 18

```
; Berechnung von Y=X!, Y soll im Speicher an Adresse adr_y
; liegen, R5 enthält bereits X (es gelte X>1)

            .text
begin:      movi    r5,6                ; Y = 6!
            mov     R6,R5
            ADDI    R6,-1
loop:       MULT    R5,R6
            ADDI    R6,-1
            BNEZ    R6,loop
            LHI     R7,adr_y>>9
            ORI     R7,adr_y&0x1FF
            S18     0(R7),R5
            TRAP    0

            .data
adr_y:      .w18    0
```

SpartanMC 18

; Ausgabe einer Zeichenkette ab dem Symbol „text1“ bis zu einer Null

```
            .text
begin:      LHI     R3,text1>>9
            ORI     R3,text1&0x1FF
loop:       L9      R1,0(R3)        ;Zeichen laden
            BEQZ    R1,ende_tx
            ADDI    R3, 1
            JALS    chout           ;Zeichen auf Konsole anzeigen
            J       loop
ende_tx:    TRAP    0

            .data
text1:      .asciiz „Hallo World\n“ ;Zeichen liegen im 9 Bit Abstand. Wird von
                                    ;text1 mit L18 geladen, steht im Rd 0x09061
                                    ;“H“ ist      0100 1000 | 0110 0001 ist “a“
                                    ;           00 1001 0000  0110 0001
```

SpartanMC 18

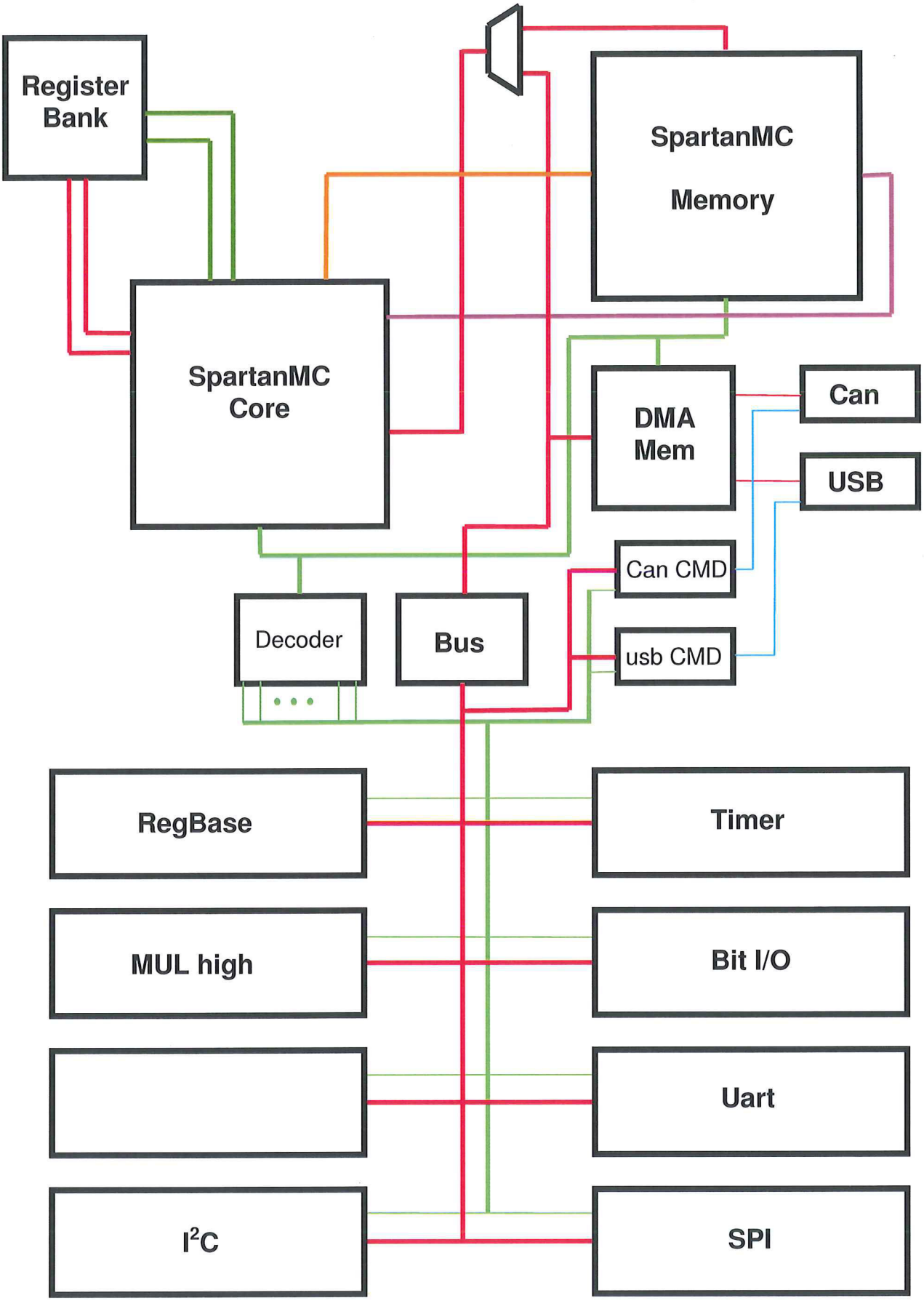
```
; 8 Bit Summe über eine Byte Liste bilden, Ende bei 0

            .text
begin:      LHI     R7,liste1>>9
            ORI     R7,liste1&0x1FF
            XOR     R2, R2          ;Summe löschen
loop:       L9      R1,0(R7)        ;Byte laden
            BEQZ    R1,ende_su
            ADDI    R7, 1           ;R7 := R7 + 1
            ADD     R2,R1           ;Summe bilden
            SGTI    R2,255          ;CC = 1 wenn größer 255
            IFADDUI R2,1            ;Übertrag addieren
            ANDI    R2,255          ;Nur 8 Bit übrig lassen
            J       loop
ende_su:    TRAP    0               ;Summe in R2

            .data
liste1:     .byte   1,2,3,4,5,6     ;Byte liegen im 9 Bit Abstand. Wird mit L18 von
            .byte   7,8,9,0         ;liste1 geladen steht im Rd 0x0202
```

30.6.2005

Das SpartanMC System



30.06.2005

IO für Spartan MC

1. IRQ

- IRQ Kontroller	für „beliebige“ Anzahl Interrups. Für jede Quelle soll eine Zieladresse einstellbar sein. Freigabe und Sperren mit CPU Befehlen EI und DI. Bei Annahme eines IRQ auf DI schalten. Priorität aller Quellen und Freigabe einstellbar.

2. Timer

Alle Einheiten benutzen den gleichen DCM
(18 Bit Startwert einstellbar).

- WDT →	Reset statt IRQ Basis Takt über **DCM** konfigurierbar. Vorteiler in 2er Potenzen in Stufen von 2^0, 2^4, 2^8 und 2^{12} einstellbar.
- 18 Bit Timer	aufwärtszählend. Basis Takt über **DCM** konfigurierbar Vorteiler in 2er Potenzen von 2^0 bis etwa 2^8 einstellbar. unabhängig vom WDT Takt! optional Capture (Konfigurierbar Flanke/Zustand IRQ). optional Compare (Konfigurierbar IRQ und Outputmodul) optional Outputmodul für alle Compare Register des Timers.
- RTI	Basis Takt über **DCM** konfigurierbar Vorteiler in 2er Potenzen von 2^0 bis etwa 2^8 einstellbar.
- Pulse Accu	Zählen von Impulsen an einem Eingang oder der RTI Impulse bis zu einem Impuls an einem Eingang.

3. Digital I/O

Input	beliebige generierbare Anzahl Inputports mit IRQ Möglichkeit einstellbar auf Flanken oder Pegel.
Output	beliebige generierbare Anzahl Outputports.
Bidirektional	Für Mem- und LCD- Datenbusse.
Input	mit Strobe – Signal, welches IRQ setzen kann. Strobe konfigurierbar auf Flanken oder Pegel.
Output	mit Strobe – Impuls für Handshake mit Input.

4. UART

- mit Modem Leitungen (konfigurierbar).
- Baud Generator über **DCM** auf maximal 115200 Baud und mit Vorteiler auf die üblichen Werte bis 75 Baud teilbar.
- Bitlänge einstellbar.
- Parität einstellbar.
- Stop Bits einstellbar. (1, 1½, 2)
- FIFO mit einstellbarer Tiefe.

5. SPI

- als Master oder Slave konfigurierbar.
- Taktfrequenz über **DCM** und Vorteiler einstellbar.
- Die Phasenlage des Taktes zu den Daten soll frei wählbar sein.
- IRQ nach n Takten einstellbar.
- Datenregister für n*8 Bit oder 9 Bit konfigurierbar.

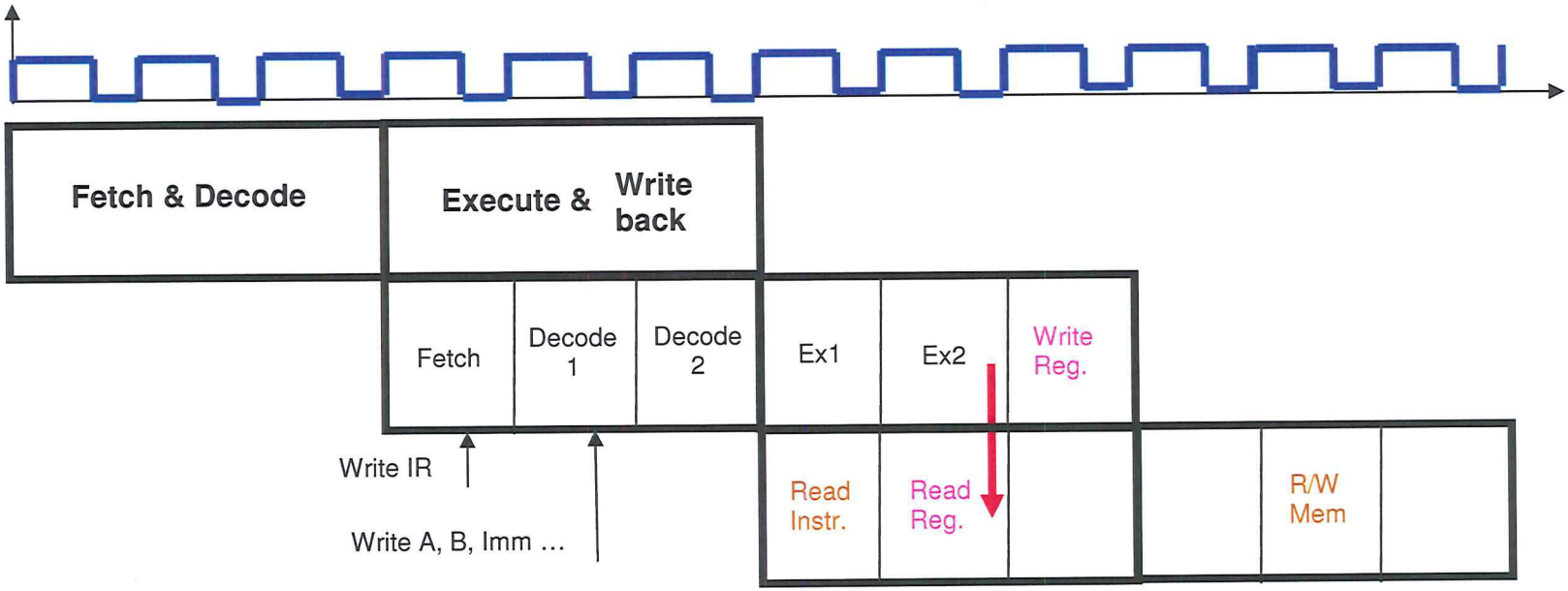
6. I^2C

- als Master oder Slave konfigurierbar.
- Sende-, Empfangsregister + Ack – Handling.
- optionale Arbitierung
- Adresserkennung für Slave.

15.07.2005

SpartanMC mit 2 stufiger Pipeline

- Ein Pipeline Takt benötigt die Zeit von mindestes 3 Speicherzyklen.
- Während dem lesen der Register kann ein anderer Befehl Daten aus dem Speicher lesen oder schreiben.
- Welche Zeit braucht die längste ALU - Operation? Lassen sich alle Operationen in der Zeit von 2 Speicherzyklen realisieren?



Bypass von Execute 2 nach Decode 1
für Register und Speicher.

Aufteilung des Befehlsablaufes auf die 6 Phasen jedes Befehls:

1. Fetch — für alle Befehlstypen gleicher Ablauf
   IR ← Mem[PC] PC_{neu} ← PC + 1 (PC hat immer Wortadressen!)

2. Decode 1 — mit vorausschauender Sprungzielberechnung und Test der Sprungbedingung für alle Befehle
   A ← GPR[$IR_{12...9}$] B ← GPR[$IR_{8...5}$] Imm ← (arit & IR_8)^9 ## $IR_{8...0}$
   Disp ← 0^{13} ## $IR_{4...0}$
   Off ← IR_{12}^5 ## $IR_{12...0}$

3. Decode 2
   Cond ← A op 0 PC_{br} ← PC_{neu} + Imm
   Con2 ← CC op 0 PC_j ← PC_{neu} + Off

4. Execute 1
   für Mem-Befehle Result ← A + Disp (nur positive Werte im Disp)
   für ALU-Befehle Result ← A op B oder Result ← A op Imm

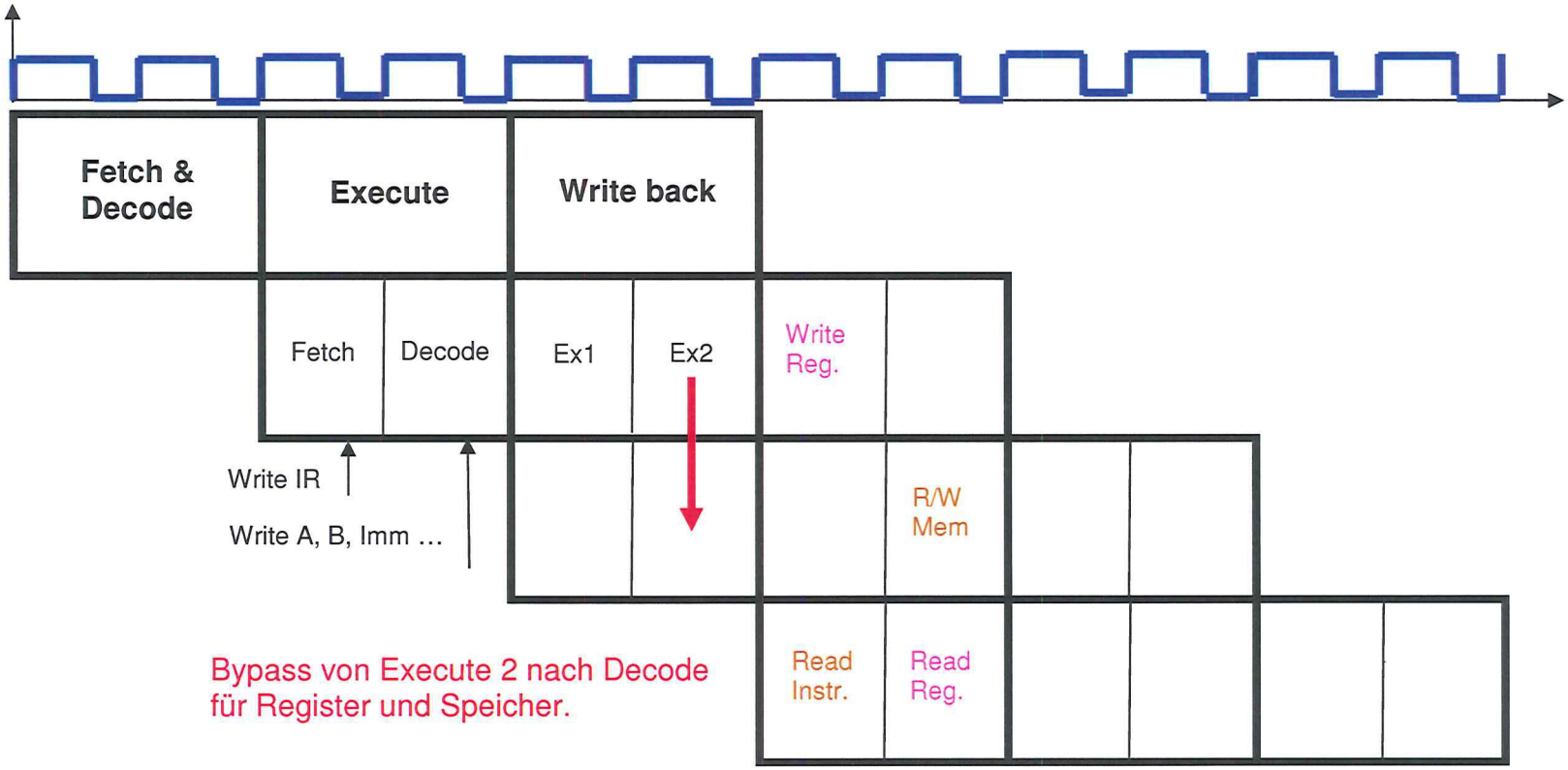
5. Execute 2
   für Mem-Befehle **LMD ← M[Result]** bei Load, und **M[Result] ← B** bei Store.
   Für ALU-Befehle wie Execute 1 wenn notwendig (MUL) sonst
   Result2 ← Result

6. Write back
   GPR[IR8…5] ← Result2 für R-Type und M-Type Befehle
   GPR[IR12…9] ← Result2 für I-Type Befehle

13.07.2005

SpartanMC mit 3 stufiger Pipeline

- Ein Pipeline Takt benötigt die Zeit von mindestes 2 Speicherzyklen.
- Während dem lesen eines Befehls kann das Ergebnis eines anderen Befehls in den Speicher oder ein Register geschrieben werden.
- Während dem lesen der Register kann ein anderer Befehl (load) Daten aus dem Speicher holen.
- Welche Zeit braucht die längste ALU - Operation? Lassen sich alle Operationen in der Zeit von 2 Speicherzyklen realisieren? Wenn nicht, dann kann eine 2 stufige Pipeline mit 3 Speicherzyklen pro Pipeline Takt sinnvoll sein.



Aufteilung des Befehlsablaufes auf die 6 Phasen jedes Befehls:

1. Fetch — für alle Befehlstypen gleicher Ablauf
   $IR \leftarrow Mem[PC]$ $PC_{neu} \leftarrow PC + 1$ (PC hat immer Wortadressen!)

2. Decode — mit vorausschauender Sprungzielberechnung und Test der Sprungbedingung für alle Befehle
   $A \leftarrow GPR[IR_{12...9}]$ $B \leftarrow GPR[IR_{8...5}]$ $Imm \leftarrow (arit \,\&\, IR_8)^9 \,\#\#\, IR_{8...0}$
   $Cond \leftarrow A \; op \; 0$ $PC_{br} \leftarrow PC_{neu} + Imm$ $Disp \leftarrow 0^{13} \,\#\#\, IR_{4...0}$
   $Con2 \leftarrow CC \; op \; 0$ $PC_j \leftarrow PC_{neu} + Off$ $Off \leftarrow IR_{12}{}^5 \,\#\#\, IR_{12...0}$

3. Execute 1 — für Mem-Befehle $Result \leftarrow A + Disp$ (nur positive Werte im Disp)
   für ALU-Befehle $Result \leftarrow A \; op \; B$ oder $Result \leftarrow A \; op \; Imm$
   für Sprungbefehle die orange dargestellten Operationen aus Decode, wenn dort keine Zeit dafür vorhanden ist.

4. Execute 2 — für Mem-Befehle **LMD ← M[Result]** bei Load, und **M[Result] ← B** bei Store.
   Für ALU-Befehle wie Execute 1 wenn notwendig (MUL) sonst
   $Result2 \leftarrow Result$

5. Write back — $GPR[IR8...5] \leftarrow Result2$ für R-Type und M-Type Befehle
   $GPR[IR12...9] \leftarrow Result2$ für I-Type Befehle

6. ungenutzt

SpartanMC 18

Inhalte:

Der SpartanMC 18 – Prozessor

- Befehlssatz
- Adressierung
- Programmierung

3. Versuch 27.05.2006

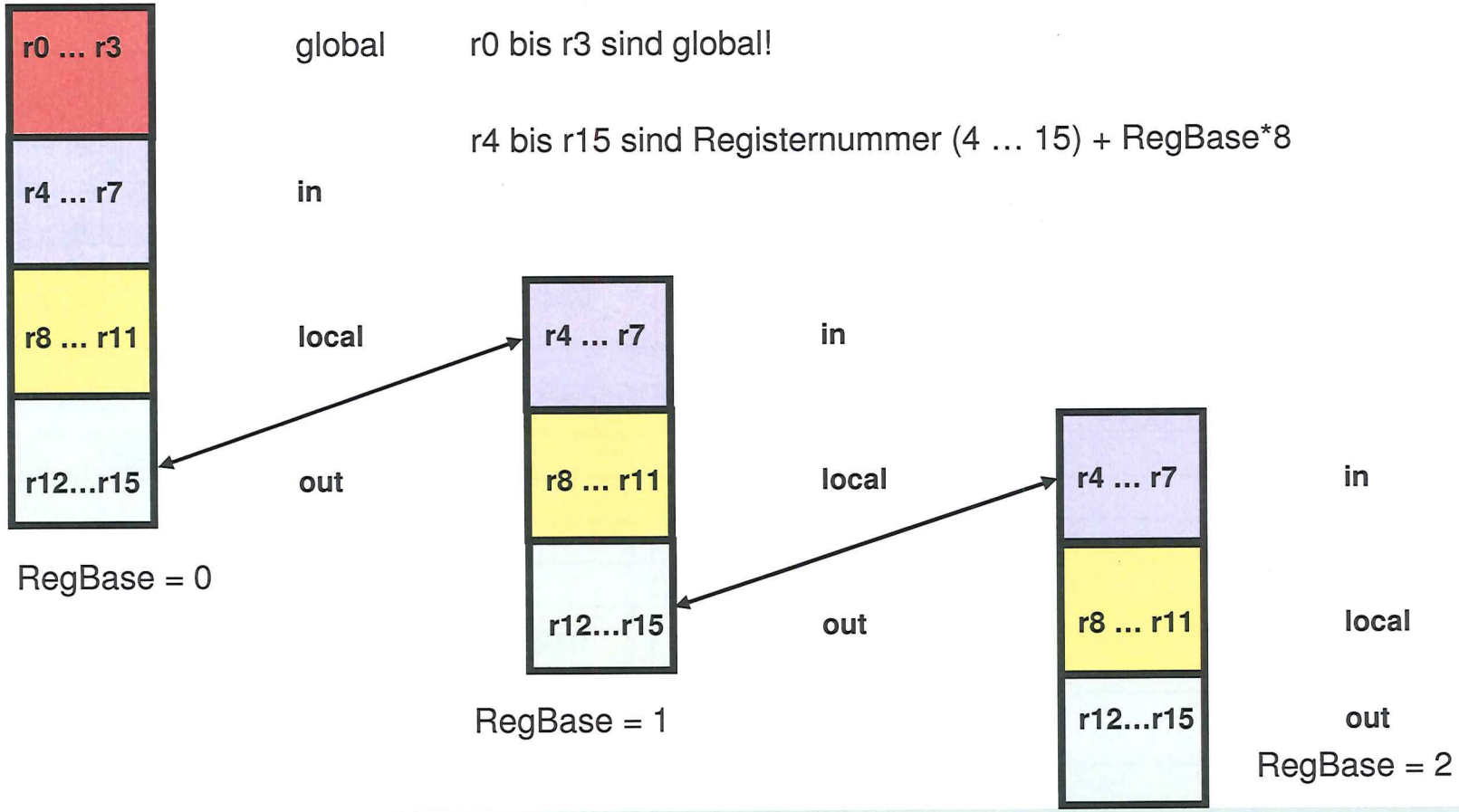
SpartanMC 18

Der SpartanMC 18 – Prozessor

- Lade/Speicher-Architektur.
- 16(1024) * 18 Bit Register r0 bis r15. r0 bis r3 sind global. R4 bis r15 werden bei UP Aufrufen mit einem Offset von jeweils 8 im Fenster verschoben.
- und ein CC, MM, INT und RegBase Register. (RegBase kann nicht im Blockram stehen).
- Befehlssatz mit 2 Adressen.
- Adressierung des Speichers mit 5 Bit Displacement.
- Vergleichs (SET) Befehle schreiben das Ergebnis in das CC Register (Bit).
- Sprünge mit 13 Bit Offset für unbedingte und für Sprünge mit Test des CC Bit.
- Sprünge mit 9 Bit Offset bei Testung eines Registers.
- Konstanten mit 9 Bit.

SpartanMC 18

Register Fenster des SpartanMC 18:



SpartanMC 18

Hauptoperationscodes des SpartanMC

$IR_{17..16}$ \ $IR_{15..13}$	000	001	010	011	100	101	110	111
00	SPEZIAL1	SPEZIAL2	J	JALS	BEQZ	BNEZ	BEQZC	BNEZC
01	ADDI	MOVI	LHI	SIGEX	ANDI	ORI	XORI	MULI
10	L9	S9	L18	S18	SLLI		SRLI	SRAI
11	SEQI	SNEI	SLTI	SGTI	SLEI	SGEI	IFADDUI	IFSUBUI

Erweiterte Operationscodes func im R-Befehlsformat des SpartanMC (SPEZIAL1 und SPEZIAL2)

$IR_{4...3}$ \ $IR_{2..0}$	000	001	010	011	100	101	110	111
00	ORCC	ANDCC	MOVI2C	MOVC2I	SLL	MOV	SRL	SRA
01	SEQU	SNEU	SLTU	SGTU	SLEU	SGEU		
10							CBITS	SBITS
11								
00	RFE	TRAP	JR	JALR	JRS	JALRS		
01								
10	ADD	ADDU	SUB	SUBU	AND	OR	XOR	MUL
11	SEQ	SNE	SLT	SGT	SLE	SGE	MOVI2S	MOVS2I

SpartanMC 18

Die Befehle entsprechen im wesentlichen den DLX Befehlen, sie haben aber nur 2 Operanden. Zur Vereinfachung der Hardware soll im Befehlskode das Zielregister immer links stehen. Die Nummer des Spezialregisters soll dagegen immer in den rechten 4 Operanden Bits des Befehlswortes im R- Format stehen, auch wenn das Spezialregister das Ziel der Operation ist.
Neue, von der DLX abweichende Befehle des SpartanMC sind:

```
BEQZC    name     ;Springe wenn der Inhalt des CC-Register (Bit) Null ist.
BNEZC    name     ;Springe wenn der Inhalt des CC-Register (Bit) nicht Null ist.

SBITS    CC       ;CC <-- 1    (CC hat im rechten Operandenfeld des Befehlskode den Wert 0)
CBITS    CC       ;CC <-- 0    (ist RESET Belegung)

MOVI2C   Rs       ;CC  <--  Rs
MOVC2I   Rd       ;Rd  <--  CC

MULI     Rd, i    ;18 Bit Multiplikation mit einem 9 Bit Immediate (oberen 18 Bit vom Resultat in Spez. Reg.)
MUL      Rd, Rs   ;18 Bit Multiplikation (oberen 18 Bit vom Resultat in Spez. Reg.)

MOVI     Rd, i    ;Rd mit i ohne Vorzeichenerweiterung laden.
MOV      Rd, Rs   ;Rd mit Inhalt von Rs laden.
```

SpartanMC 18

Befehle zur Verarbeitung von Daten im 9(8) Bit Format:

```
L9      Rd, disp(Rs)      ;Lädt die unteren 9 Bit für ungerade Adressen und bei geraden die oberen
                          ;9 Bit. In Rd stehen die 9 Bit mit „0“ erweitert
S9      disp(Rs1), Rs2    ;speichert in die unteren 9 Bit für ungerade Adressen und bei geraden in
                          ;die oberen 9 Bit.
```

(Die Befehle arbeiten also im Big Endien. Zur Adressierung der oberen 128K 18 Bit Worte muss mm = 1 gesetzt werden.)

```
SBITS   MM        ;mm <-- 1    (mm hat im rechten Operandenfeld des Befehlskode den Wert 1)
CBITS   MM        ;mm <-- 0    (ist RESET Belegung)

SIGEX   Rd, Imm   ;Vorzeichen Erweiterung ab Bitnummer in Imm. (Typisch sind 7, 8 und 15)

IFADDUI Rd, Imm   ;IF (CC != 0)  then    Rd <-- Rd + Imm
IFSUBUI Rd, Imm   ;IF (CC != 0)  then    Rd <-- Rd – Imm

SBITS   INT       ;INT <-- 1   Interrupt Freigabe
CBITS   INT       ;INT <-- 0   Interrupt Sperren (ist RESET Belegung)
                  ;            (INT hat im rechten Operandenfeld des Befehlskode den Wert 2)
```

SpartanMC 18

Unterprogrammbefehle mit Sichern der Register (r11 ist letztes lokales Register zum Retten des PC)

```
JALRS    r4      ;PC <-- R4,      RegBase <-- RegBase + 1 ,    R11 <-- PC + 1
JALS     label   ;PC <-- label,   RegBase <-- RegBase + 1 ,    R11 <-- PC + 1
JRS      r11     ;PC <-- R11,     RegBase <-- RegBase – 1
```

Unterprogrammbefehl ohne Sichern der Register (r11 ist letztes lokales Register zum Retten des PC)

```
JALR     r5      ;PC <-- R5,      R11 <-- PC + 1,
JR       r11     ;PC <-- R11
```

SpartanMC 18

Die 4 SpartanMC- Befehlsformate sollen an Hand der folgenden Befehle in die 18-Bit-Befehlswörter codiert werden.

- ALU-Befehle

```
add     r2, r1
andi    r4, 0xff
seq     r5, r6          ;Setze CC-Register
lhi     r14, 0xff       ;  9 Bit nach High von r14 laden
```

- Verzweigebefehle

```
bnez    r3, loop        ;  9 Bit Offset
beqzc   m0001           ;13 Bit Offset,  testet CC-Register
j       addr1           ;13 Bit Offset
trap    18              ;wie jals zur Adresse 18 (oder 0 ... 255)
```

- Lade/Speicher-Befehle

```
l9      r5, 0x2(r4)
```

- Steuerbefehle

```
cbits   0               ;CC löschen
sbits   2               ;INT setzen (Freigeben der Interrupts)
```

- Transportbefehle

```
movi2s  1, r4           ;Wert von r4 an 7 Segment Anzeige
                        ;senden.
```

SpartanMC 18

R-Type (Register)

17 13	12 9	8 5	4 0
Operationscode opc	R e g i s t e r rd/rs2	 rs1	 func
00 001	0010	0001	10 000
00 001	0110	0101	11 000
00 001	0100	0001	11 110

IR

add

Seq

movi2s

ALU-Befehl add r2,r1 ; r2 ⟵ r2 + r1

seq r5,r6 ; cc ⟵ r5 == r6

Transportbefehl movi2s 1,r4 ; LED[6:0] ⟵ r4

SpartanMC 18

R-Type (Register)

Steuerbefehle

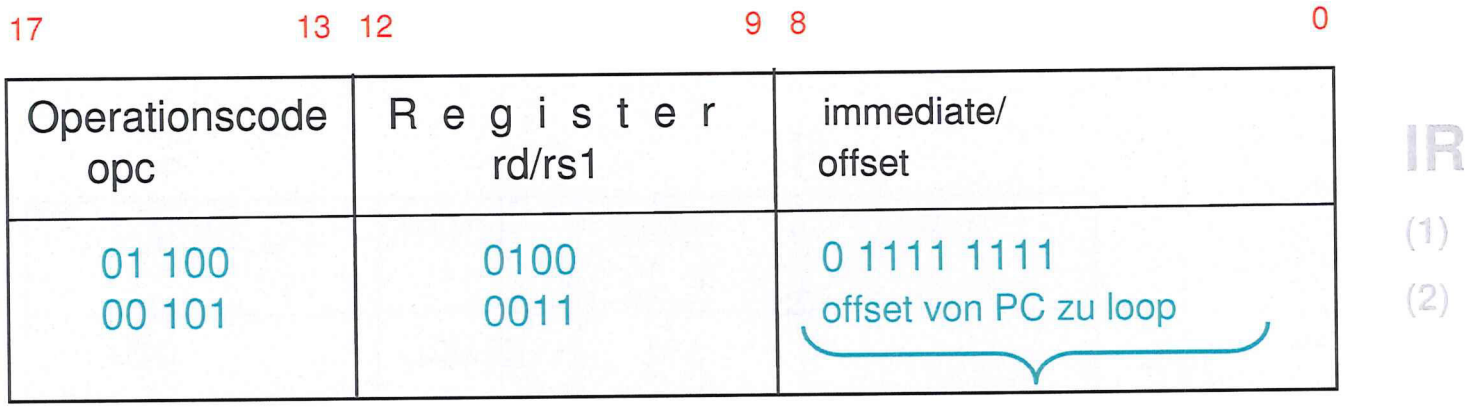
```
cbits   0      ; cc  <---  0
sbits   2      ; int <---  1
```

Verzweigebefehl

```
trap    18     ; PC  <---  18
               ; RegBase = Regbase+1, r11 <--- PC+1
```

SpartanMC 18

I-Type (Immediate)



(1) ALU-Befehl `andi r4,0xff` ; r4 ⟵ r4 & 0x000ff

(2) Verzweigebefehl `bnez r3,loop` ; PC ⟵ PC + (IR_8^9 ## $IR_{8...0}$)
; offset von PC+1 zu loop (mit VZ)

SpartanMC 18

M-Type (Memory)

17 13	12 9	8 5	4 0
Operationscode opc	Register rd/rs2	Register rs1	displacement
10 000	0101	0100	0 0010

IR

Lade/Speicher-Befehl l9 r5,0x2(r4) ; r5 ← 0^9 ## M[r4 +0x2]

SpartanMC 18

J-Type (Jump)

17 — Operationscode opc	13 12 — jump-offset	0
00 010	13-bit-offset von PC zu addr1	(1)
00 110	13-bit-offset von PC zu m0001	(2)

IR

(1) Verzweigebefehl `j addr1` ; PC ⟵ PC + (IR_{12}^{5} ## $IR_{12...0}$)

(2) `beqzc m0001` ; PC ⟵ PC + (IR_{12}^{5} ## $IR_{12...0}$), wenn cc = 0

; offset von PC+1 zu m0001 (mit VZ)

SpartanMC 18

Für die folgende Aufgabe soll nun eine Befehlsfolge ermittelt werden.

a = b + c;
d = e – f; a,b,c,d,e,f sind 18-Bit Worte im Speicher mit den Labels A,B,C,D,E,F

Der nachstehend angegebenen Rahmen soll dafür angewendet werden.

```
        .text       0x0
start:  lhi         r1, 0x210>>9        ;
        ...........                     ;   Laden Adresse A nach R1
        ...........                     ;
        ...........

stop:   j           stop                ; Endlosschleife

        .data       0x210
A:      .w18        0x.....
B:      .w18        0x.....
C:      .w18        0x.....
D:      .w18        0x.....
E:      .w18        0x.....
F:      .w18        0x.....
```

SpartanMC 18

```
        .text     0x0
start:  lhi       r1, A>>9          ;
        ori       r1, A&0x1ff       ;   Laden Adresse A = 0x210 nach r1
        l18       r2, 2(r1)         ; Laden b nach r2
        l18       r3, 4(r1)         ; Laden c nach r3
        add       r3, r2
        s18       0(r1), r3         ; a = b + c
        l18       r2, 8(r1)
        l18       r3,10(r1)
        sub       r2, r3
        s18       6(r1), r2
stop:   j         stop              ; Endlosschleife

        .data     0x210
A:      .w18      0x0               ; 3 + 5 = 8
B:      .w18      0x3
C:      .w18      0x5
D:      .w18      0x0               ; 9 – 6 = 3
E:      .w18      0x9
F:      .w18      0x6
```

SpartanMC 18

Zwei gleich große Datenfelder mit 18-Bit-Daten der Länge n sollen auf Gleichheit untersucht werden.

```
                .text       0
start:


                .data       0x200
adr_F1:         .w18                    ; 1. Adresse von Feld1
                ..........
adr_F2:         .w18                    ; 1. Adresse von Feld2

                .........
adr_n:          .byte                   ; Länge der Felder F1 und F2
ergebnis:       .byte       0           ; Ergebnis: 0 für ≠ , 1 für =
```

SpartanMC 18

```
            .text     0x0
start:      lhi       r5, adr_F1>>9          ; die oberen 9 Bit von F1 in r5 laden.
            ori       r5, adr_F1&0x1ff       ; Adr. von F1 in r5
            lhi       r6, adr_F2>>9
            ori       r6, adr_F2&0x1ff       ; Adr. von F2 in r6
            lhi       r7, adr_n>>9
            ori       r7, adr_n&0x1ff
            l9        r7, 0(r7)              ; Länge der Felder in r7, Zähler für Test
            lhi       r8, ergebnis>>9
            ori       r8, ergebnis&0x1ff
            xor       r11, r11               ; clr r11
loop:       l18       r9, 0(r5)
            l18       r10, 0(r6)
            seq       r9, r10                ; cc = 1 wenn gleich, sonst cc = 0
            addi      r5, 0x1
            addi      r6, 0x1
            addi      r7, -1
            beqzc     store                  ; unterschiedliche Worte, Vergleichen beenden
            bnez      r7, loop
            addi      r11, 1                 ; alle gleich → r11 = 1
store:      s9        0(r8), r11
stop:       j         stop
```

SpartanMC 18

```
            .data                   ; Datenbereich

adr_F1:     .w18        1           ; 1. Adresse von Feld1
            .w18        2
            ..........
adr_F2:     .w18        1           ; 1. Adresse von Feld2
            .w18        2

            .........

lae         equ         (adr_F2 – adr_F1)/2

adr_n:      .byte       lae         ; Länge der Felder F1 und F1

ergebnis:   .byte       0           ; Ergebnis: 0 für ≠ , 1 für =
```

SpartanMC 18

; Berechnung von Y=X!, Y soll im Speicher an Adresse adr_y
; liegen, R5 enthält bereits X (es gelte X>1)

```
            .text     0
begin:      movi      r5, 9              ; Y = 9!
            lhi       r8, 0x100          ; Maske für Bit 17 in r5
            mov       r6, r5
            addi      r6, -1
loop:       mul       r5, r6
            movs2i    1, r4              ; SFR 1 = HIGH von MUL nach r4
            addi      r6, -1
            mov       r9, r5
            and       r9, r8
            bnez      r4, zugross
            bnez      r9, zugross        ; Vorzeichenbit ist gesetzt, Zahl würde Negativ
            bnez      r6, loop
mem:        lhi       r7, adr_y>>9
            ori       r7, adr_y&0x1FF
            s18       0(r7), r5
stop:       j         stop
zugross:    xor       r5, r5             ; 0 kann X! nicht werden.
            j         mem

            .data     0x200
adr_y:      .w18      0
```

SpartanMC 18

```
; Ausgabe einer Zeichenkette ab dem Symbol „text1“ bis zu einer Null

            .text
begin:      LHI     R3,text1>>9
            ORI     R3,text1&0x1FF
loop:       L9      R1,0(R3)        ;Zeichen laden
            BEQZ    R1,ende_tx
            ADDI    R3, 1
            JALS    chout           ;Zeichen auf Konsole anzeigen
            J       loop
ende_tx:    TRAP    0

            .data
text1:      .asciiz „Hallo World\n“ ;Zeichen liegen im 9 Bit Abstand. Wird von
                                    ;text1 mit L18 geladen, steht im Rd 0x09061
                                    ;“H“ ist      0100 1000 | 0110 0001 ist “a“
                                    ;           00 1001 0000  0110 0001
```

SpartanMC 18

```
; 8 Bit Summe über eine Byte Liste bilden, Ende bei 0

            .text
begin:      LHI     R7,liste1>>9
            ORI     R7,liste1&0x1FF
            XOR     R2, R2              ;Summe löschen
loop:       L9      R1,0(R7)            ;Byte laden
            ADDI    R7, 1               ;R7 := R7 + 1
            BEQZ    R1,ende_su
            ADD     R2,R1               ;Summe bilden
            SGTI    R2,255              ;CC = 1 wenn größer 255
            ANDI    R2,255              ;Nur 8 Bit übrig lassen
            IFADDUI R2,1                ;8 Bit Übertrag addieren
            J       loop
ende_su:    J       ende_su             ;Summe in R2

            .data
liste1:     .byte   1,2,3,4,5,6         ;Byte liegen im 9 Bit Abstand. Wird mit L18 von
            .byte   7,8,9,250,10,0      ;liste1 geladen steht im Rd 0x0202
```

SpartanMC 18

Inhalte:

Der SpartanMC 18 – Prozessor

- Registerstruktur
- Pipeline
- Befehlssatz

27.05.2006

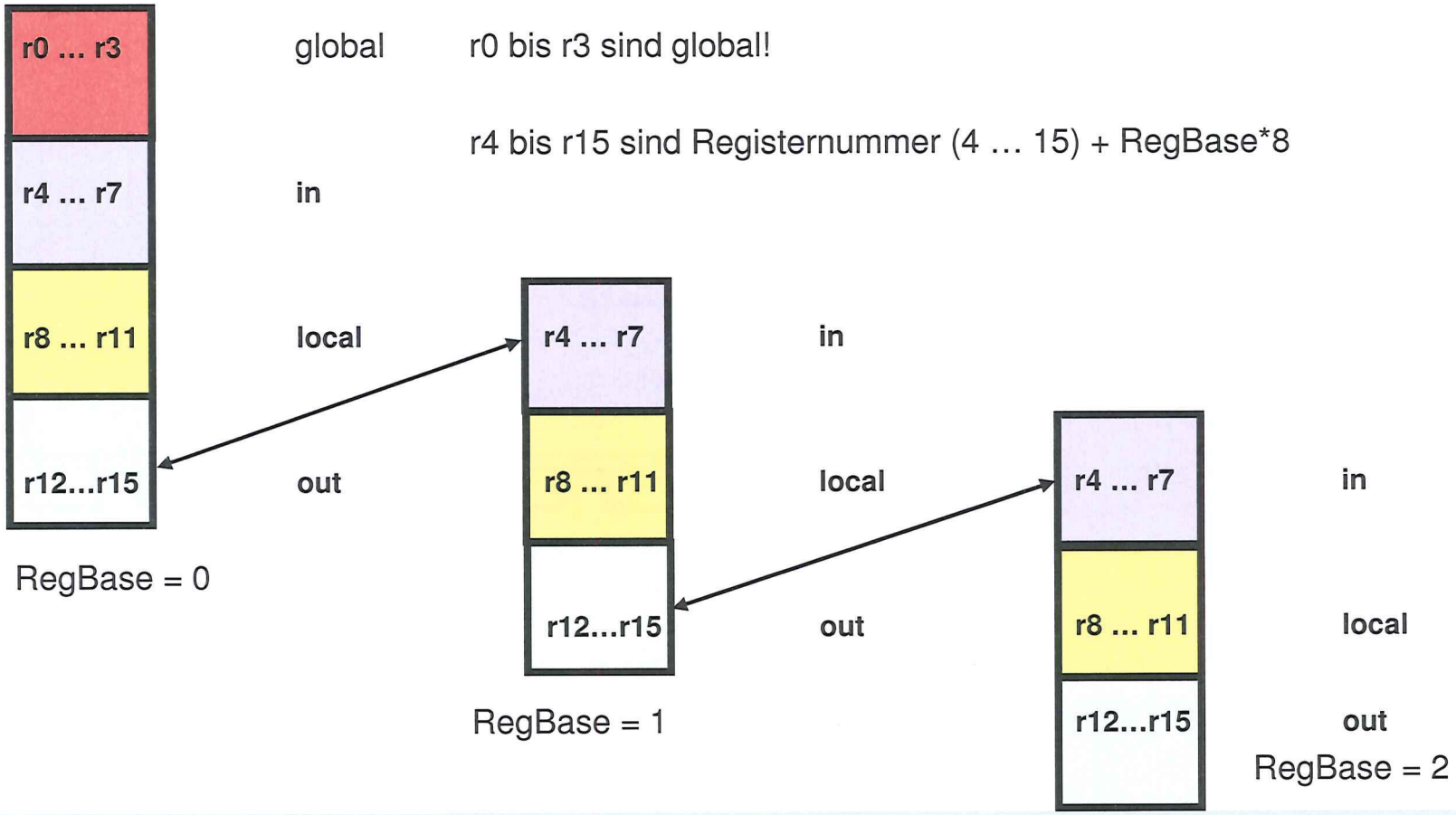
SpartanMC 18

Der SpartanMC 18 – Prozessor

- Lade/Speicher-Architektur
- 16(1025) * 18 Bit Register r0 bis r15. r0 bis r3 sind global. R4 bis r15 werden bei UP Aufrufen mit einem Offset von jeweils 8 im Fenster verschoben.
- und ein CC, MM, EI und RegBase Register. (RegBase kann nicht im Blockram stehen)
- Befehlssatz mit 2 Adressen.
- Adressierung des Speichers mit 5 Bit Displacement.
- Vergleichs (SET) Befehle schreiben das Ergebnis in das CC Register (Bit).
- Sprünge mit 13 Bit Offset für unbedingte und für Sprünge mit Test des CC Bit.
- Sprünge mit 9 Bit Offset bei Testung eines Registers.
- Konstanten mit 9 Bit.

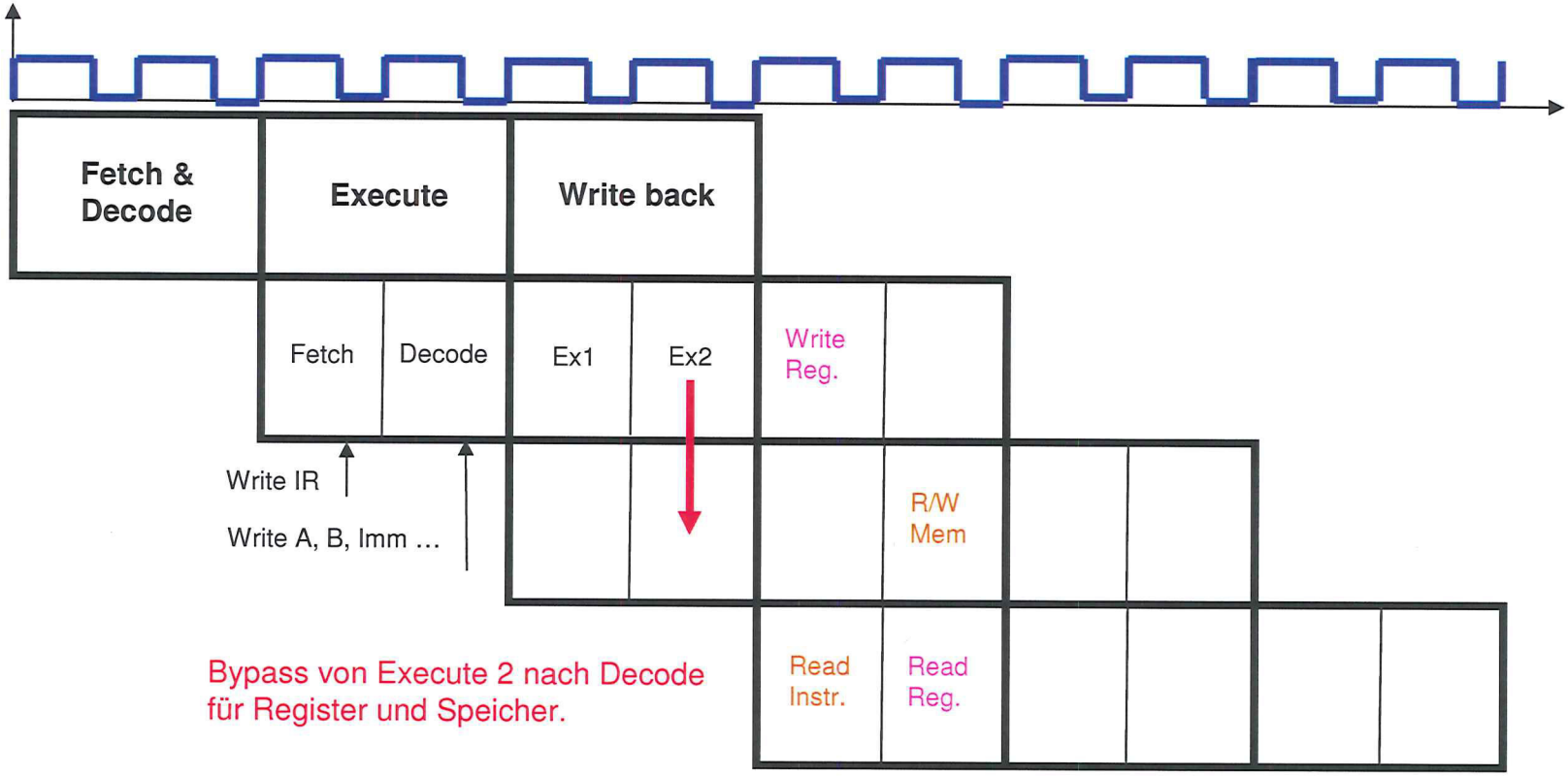
SpartanMC 18

Register Fenster des SpartanMC 18:



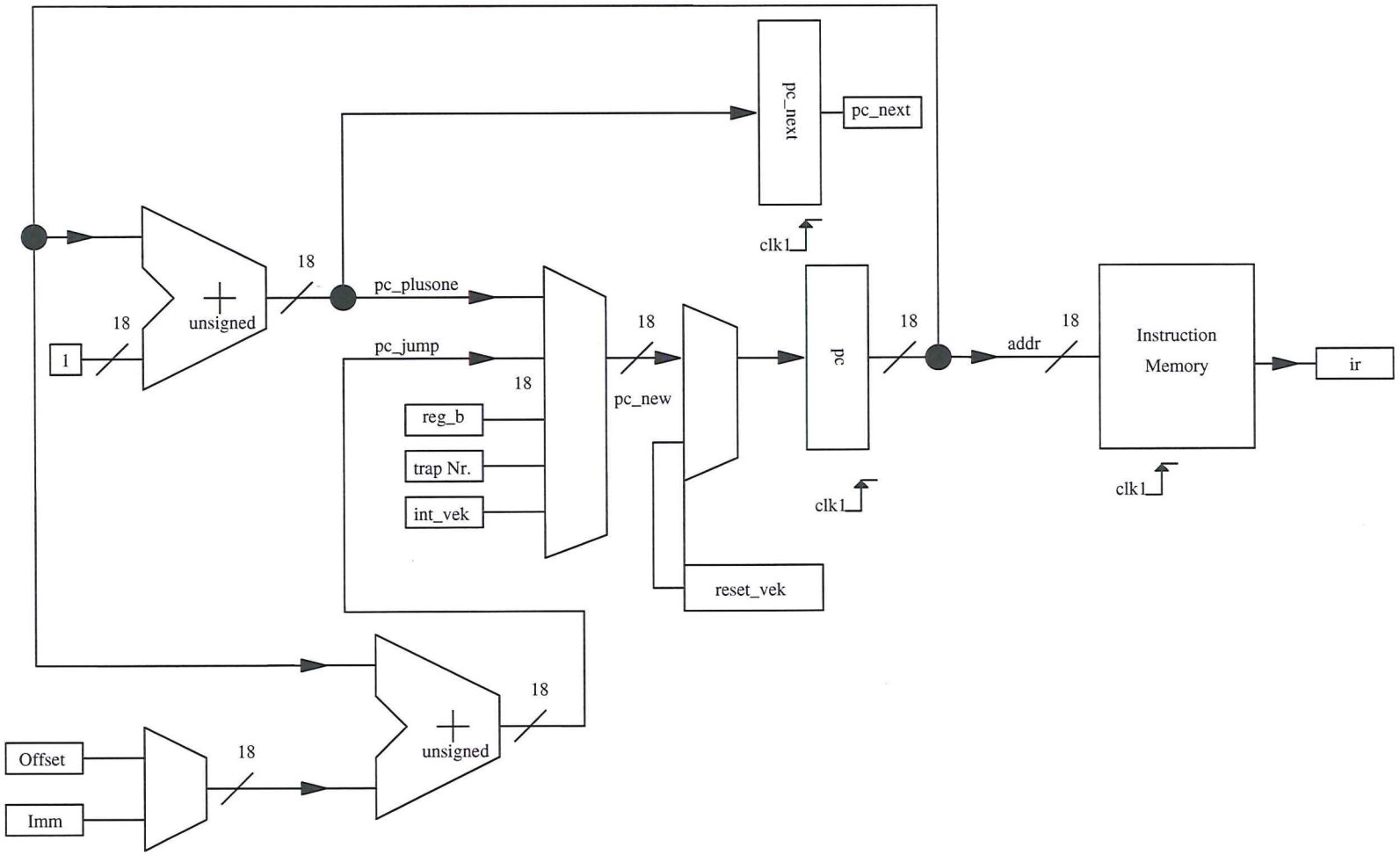
SpartanMC 18

3 stufige Pipeline des SpartanMC



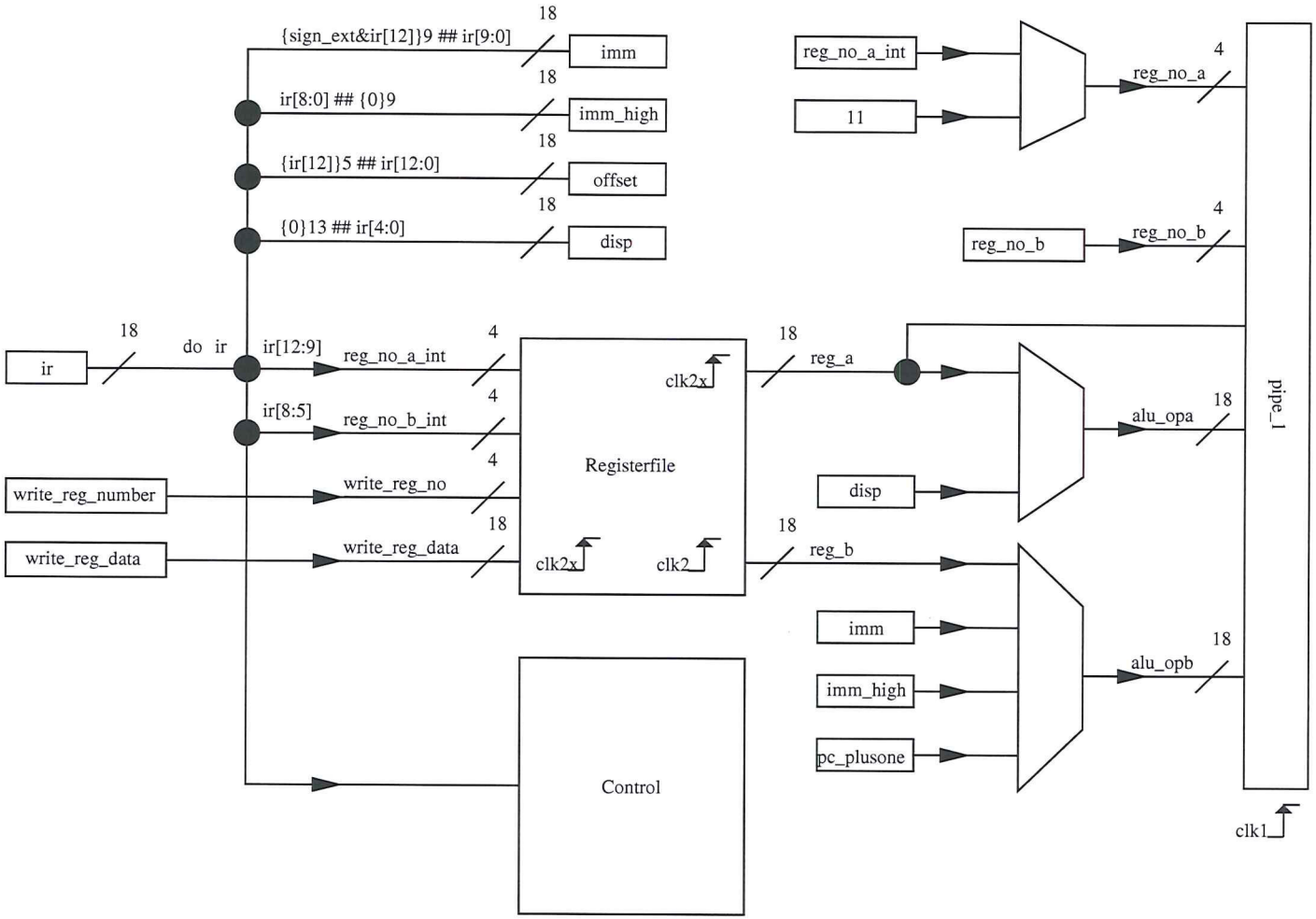
SpartanMC 18

Fetch

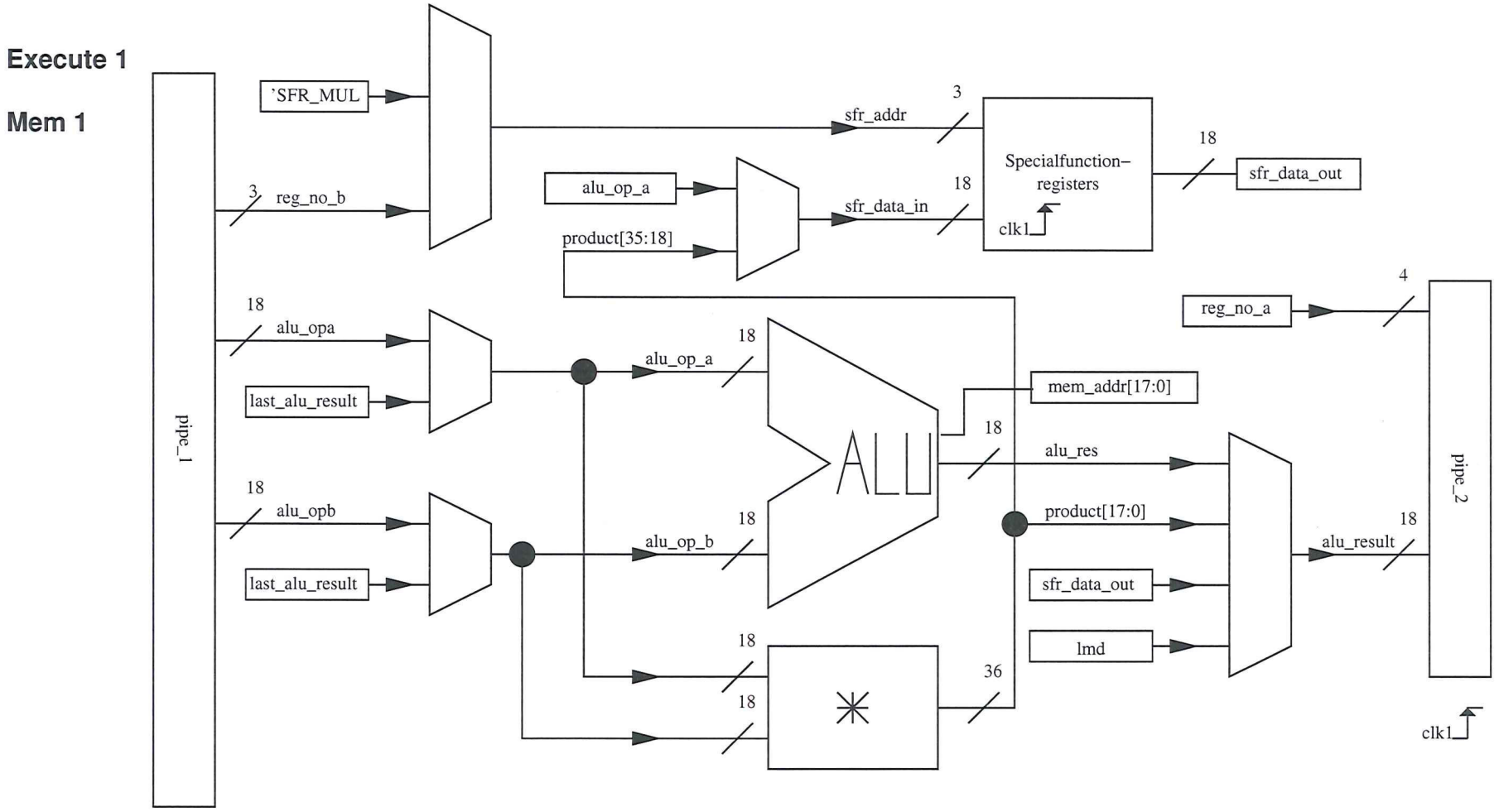


SpartanMC 18

Decode

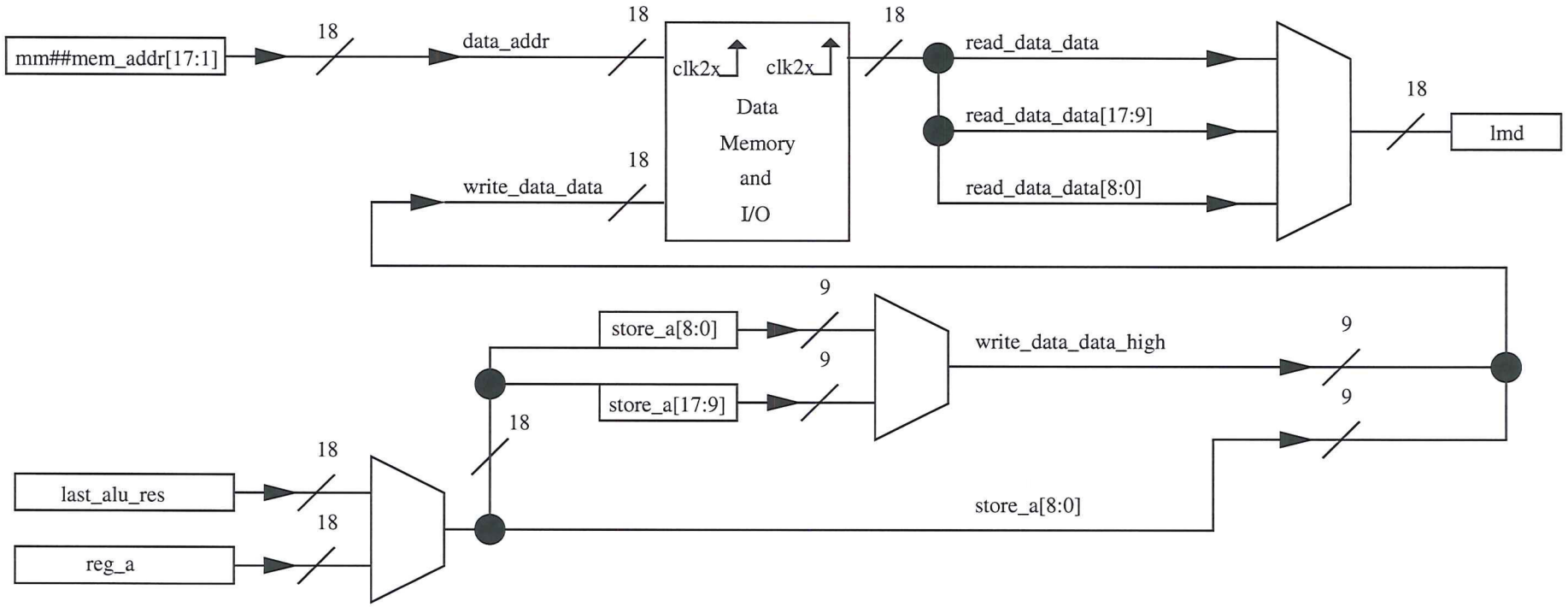


SpartanMC 18

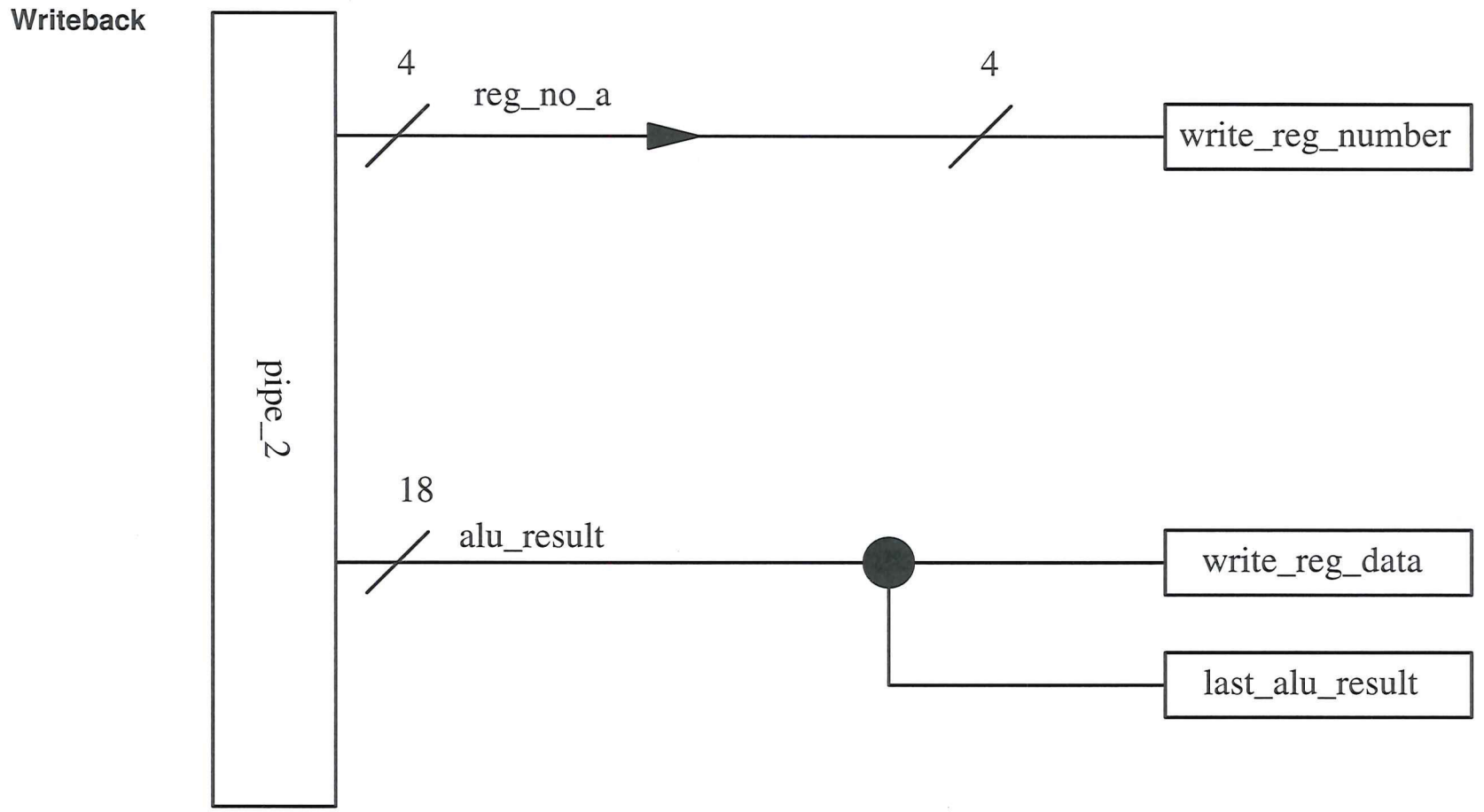


SpartanMC 18

Execute 2 Mem 2



SpartanMC 18



SpartanMC 18

Hauptoperationscodes des SpartanMC

$IR_{17..16}$ \ $IR_{15..13}$	000	001	010	011	100	101	110	111
00	SPEZIAL1	SPEZIAL2	J	JALS	BEQZ	BNEZ	BEQZC	BNEZC
01	ADDI	MOVI	LHI	SIGEX	ANDI	ORI	XORI	MULI
10	L9	S9	L18	S18	SLLI		SRLI	SRAI
11	SEQI	SNEI	SLTI	SGTI	SLEI	SGEI	IFADDUI	IFSUBUI

Erweiterte Operationscodes func im R-Befehlsformat des SpartanMC (SPEZIAL1 und SPEZIAL2)

$IR_{4...3}$ \ $IR_{2..0}$	000	001	010	011	100	101	110	111
00	ORCC	ANDCC	MOVI2C	MOVC2I	SLL	MOV	SRL	SRA
01	SEQU	SNEU	SLTU	SGTU	SLEU	SGEU		
10							CBITS	SBITS
11								
00	RFE	TRAP	JR	JALR	JRS	JALRS		
01								
10	ADD	ADDU	SUB	SUBU	AND	OR	XOR	MUL
11	SEQ	SNE	SLT	SGT	SLE	SGE	MOVI2S	MOVS2I

SpartanMC 18

Inhalte:

Der SpartanMC 18 – Prozessor

- Befehlssatz
- Adressierung
- Programmierung

4. Versuch 13.10.2010

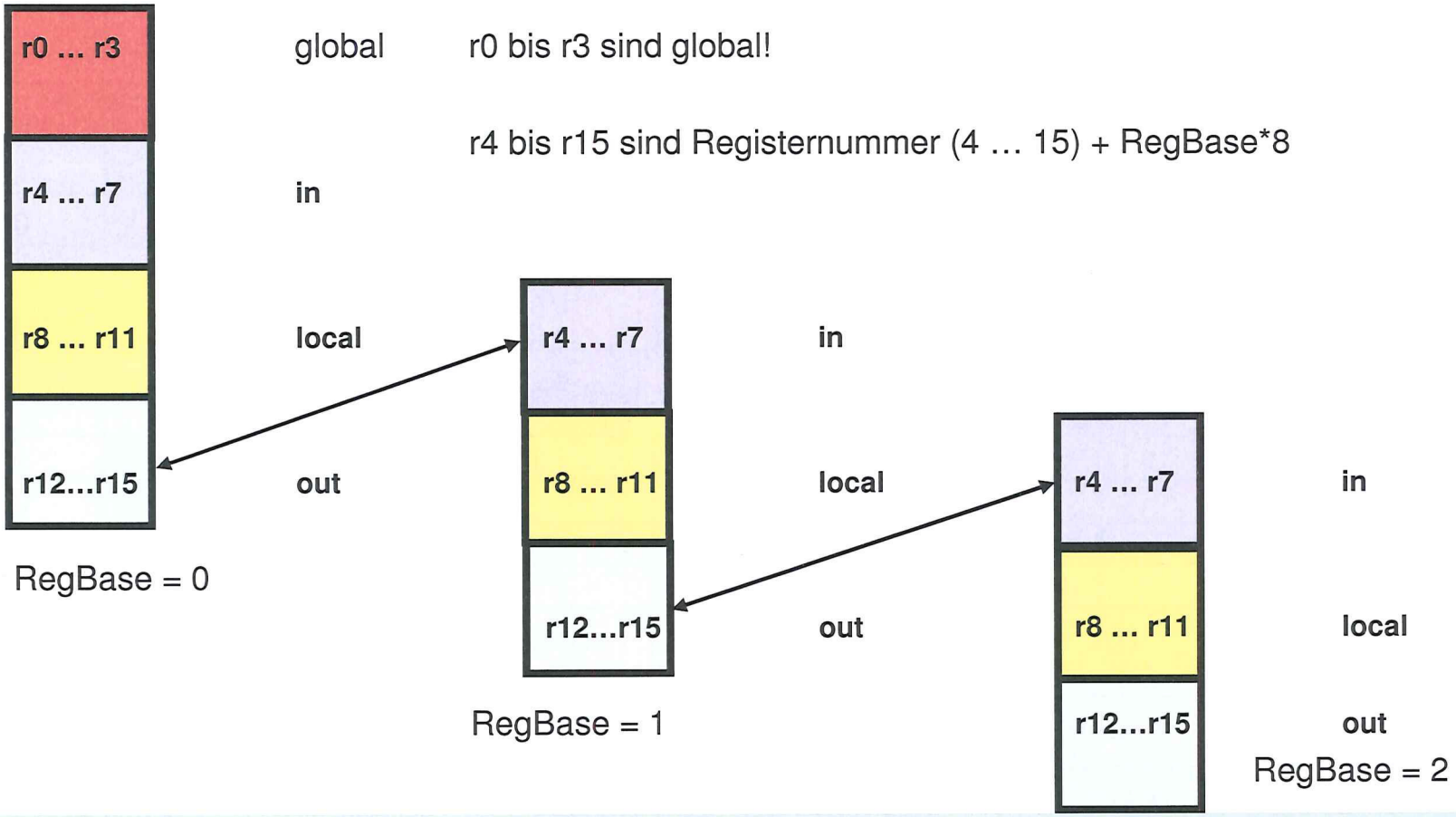
SpartanMC 18

Der SpartanMC 18 – Prozessor

- Lade/Speicher-Architektur.
- 16(1024) * 18 Bit Universalregister r0 bis r15. r0 bis r3 sind global. R4 bis r15 werden bei UP Aufrufen mit einem Offset von jeweils 8 im Fenster verschoben.
- und ein CC, MM, INT und RegBase Register. (Sind nicht im Blockram).
- Befehlssatz mit 2 Adressen.
- Adressierung des Speichers mit 5 Bit Displacement.
- Vergleichs (SET) Befehle schreiben das Ergebnis in das CC Register (Bit).
- Sprünge mit 13 Bit Offset für unbedingte und für Sprünge mit Test des CC Bit.
- Sprünge mit 9 Bit Offset bei Testung eines Registers.
- Konstanten mit 9 Bit.

SpartanMC 18

Register Fenster des SpartanMC 18:



SpartanMC 18

Hauptoperationscodes des SpartanMC (Befehl implementiert, wie bei DLX und nicht implementiert)

$IR_{17..16}$ \ $IR_{15..13}$	000	001	010	011	100	101	110	111
00	SPEZIAL1	SPEZIAL2	J	JALS	BEQZ	BNEZ	BEQZC	BNEZC
01	ADDI	MOVI	LHI	SIGEX	ANDI	ORI	XORI	MULI
10	L9	S9	L18	S18	SLLI		SRLI	SRAI
11	SEQI	SNEI	SLTI	SGTI	SLEI	SGEI	IFADDUI	IFSUBUI

Erweiterte Operationscodes func im R-Befehlsformat des SpartanMC (SPEZIAL1 und SPEZIAL2)

$IR_{4...3}$ \ $IR_{2..0}$	000	001	010	011	100	101	110	111
00	ORCC	ANDCC			SLL	MOV	SRL	SRA
01	SEQU	SNEU	SLTU	SGTU	SLEU	SGEU		
10							CBITS	SBITS
11								NOT
00	RFE	TRAP	JR	JALR	JRS	JALRS		
01								
10	ADD	ADDU	SUB	SUBU	AND	OR	XOR	MUL
11	SEQ	SNE	SLT	SGT	SLE	SGE	MOVI2S	MOVS2I

SpartanMC 18

Die Befehle entsprechen im wesentlichen den DLX Befehlen, sie haben aber nur 2 Operanden. Zur Vereinfachung der Hardware soll im Befehlskode das Zielregister immer links stehen. Die Nummer des Spezialregisters soll dagegen immer in den rechten 4 Operanden Bits des Befehlswortes im R- Format stehen, auch wenn das Spezialregister das Ziel der Operation ist.
Neue, von der DLX abweichende Befehle des SpartanMC sind:

```
BEQZC    name     ;Springe wenn der Inhalt des CC-Register (Bit) Null ist.
BNEZC    name     ;Springe wenn der Inhalt des CC-Register (Bit) nicht Null ist.

SBITS    CC       ;CC <-- 1    (CC hat im rechten Operandenfeld des Befehlskode den Wert 0)
CBITS    CC       ;CC <-- 0    (ist RESET Belegung)

MOVI2C   Rs       ;CC  <--  Rs
MOVC2I   Rd       ;Rd  <--  CC

MULI     Rd, i    ;18 Bit Multiplikation mit einem 9 Bit Immediate (oberen 18 Bit vom Resultat in Spez. Reg.)
MUL      Rd, Rs   ;18 Bit Multiplikation (oberen 18 Bit vom Resultat in Spez. Reg.)

MOVI     Rd, i    ;Rd mit i ohne Vorzeichenerweiterung laden.
MOV      Rd, Rs   ;Rd mit Inhalt von Rs laden.
```

SpartanMC 18

Befehle zur Verarbeitung von Daten im 9(8) Bit Format:

```
L9      Rd, disp(Rs)      ;Lädt die unteren 9 Bit für ungerade Adressen und bei geraden die oberen
                          ;9 Bit. In Rd stehen die 9 Bit mit „0“ erweitert
S9      disp(Rs1), Rs2    ;speichert in die unteren 9 Bit für ungerade Adressen und bei geraden in
                          ;die oberen 9 Bit.
```

(Die Befehle arbeiten also im Big Endien. Zur Adressierung der oberen 128K 18 Bit Worte muss mm = 1 gesetzt werden.)

```
SBITS   MM        ;mm <-- 1    (mm hat im rechten Operandenfeld des Befehlskode den Wert 1)
CBITS   MM        ;mm <-- 0    (ist RESET Belegung)

SIGEX   Rd, Imm   ;Vorzeichen Erweiterung ab Bitnummer in Imm. (Typisch sind 7, 8 und 15)

IFADDUI Rd, Imm   ;IF (CC != 0)  then    Rd <-- Rd + Imm
IFSUBUI Rd, Imm   ;IF (CC != 0)  then    Rd <-- Rd – Imm

SBITS   INT       ;INT <-- 1   Interrupt Freigabe
CBITS   INT       ;INT <-- 0   Interrupt Sperren (ist RESET Belegung)
                  ;            (INT hat im rechten Operandenfeld des Befehlskode den Wert 2)
```

SpartanMC 18

Unterprogrammbefehle mit Schieben des Registerfenster
(r11 ist letztes lokales Register zum Retten des PC)

```
JALRS    r4       ; RegBase <-- RegBase + 1 ,    R11 <-- PC + 1,    PC <-- R4
JALS     label    ; RegBase <-- RegBase + 1 ,    R11 <-- PC + 1,    PC <-- label
JRS      r11      ; PC <-- R11,                  RegBase <-- RegBase – 1
```

Unterprogrammbefehl ohne Schieben des Registerfenster
(r11 ist letztes lokales Register zum Retten des PC)

```
JALR     r5       ; R11 <-- PC + 1,              PC <-- R5
JR       r11      ; PC <-- R11
```

SpartanMC 18

Die 4 SpartanMC- Befehlsformate sollen an Hand der folgenden Befehle in die 18-Bit-Befehlswörter codiert werden.

- ALU-Befehle

```
add   r2, r1      ;CC = 1 bei Overflow
andi  r4, 0xff
seq   r5, r6      ;Setze CC-Register
lhi   r14, 0xff   ;  9 Bit nach High von r14 laden
```

- Verzweigebefehle

```
bnez  r3, loop    ;  9 Bit Offset
beqzc m0001       ;13 Bit Offset,  testet CC-Register
j     addr1       ;13 Bit Offset
trap  18          ;wie jals zur Adresse 18 (oder 0 ... 255)
```

- Lade/Speicher-Befehle

```
l9    r5, 0x2(r4)
```

- Steuerbefehle

```
cbits 0           ;CC löschen
sbits 2           ;INT setzen (Freigeben der Interrupts)
```

- Transportbefehle

```
movi2s 1, r4      ;Wert von r4 an 7 Segment Anzeige
                  ;senden.
```

SpartanMC 18

R-Type (Register)

17 13	12 9	8 5	4 0
Operationscode opc	Register rd/rs2	Register rs1	func
00 001	0010	0001	10 000
00 001	0110	0101	11 000
00 001	0100	0001	11 110

IR

add

Seq

movi2s

ALU-Befehl

```
add     r2,r1     ; r2  <---  r2 + r1
seq     r5,r6     ; cc  <---  r5 == r6
```

Transportbefehl

```
movi2s  1,r4      ; LED[6:0]  <---  r4
```

SpartanMC 18

R-Type (Register)

17 13	12 9	8 5	4 0
Operationscode opc	Register rd/rs2	Register rs1	func
00 000	0000	0000	10 110
00 000	0000	0010	10 111
00 001	0001	0010	00 001

IR

cbits

sbits

trap

Steuerbefehle

```
cbits   0    ; cc  <---  0
sbits   2    ; int <---  1
```

Verzweigebefehl

```
trap    18   ; PC  <---  18
             ; RegBase = Regbase+1, r11 <---  PC+1
```

SpartanMC 18

I-Type (Immediate)

17 13	12 9	8 0
Operationscode opc	R e g i s t e r rd/rs1	immediate/ offset
01 100 00 101	0100 0011	0 1111 1111 offset von PC zu loop

IR
(1)
(2)

(1) ALU-Befehl `andi r4,0xff` ; r4 ⟵ r4 & 0x000ff

(2) Verzweigebefehl `bnez r3,loop` ; PC ⟵ PC + (IR_8^9 ## $IR_{8...0}$)
; offset von PC+1 zu loop (mit VZ)

SpartanMC 18

M-Type (Memory)

17 … 13	12 … 9	8 … 5	4 … 0
Operationscode opc	Register rd/rs2	Register rs1	displacement
10 000	0101	0100	0 0010

IR

Lade/Speicher-Befehl l9 r5,0x2(r4) ; r5 ⟵ 0^9 ## M[r4 +0x2]

SpartanMC 18

J-Type (Jump)

17 … 13	12 … 0
Operationscode opc	jump-offset
00 010	13-bit-offset von PC zu addr1 (1)
00 110	13-bit-offset von PC zu m0001 (2)

IR

(1) Verzweigebefehl `j addr1` ; PC ⟵ PC + (IR_{12}^{5} ## $IR_{12...0}$)

(2) `beqzc m0001` ; PC ⟵ PC + (IR_{12}^{5} ## $IR_{12...0}$), wenn cc = 0

; offset von PC+1 zu m0001 (mit VZ)

SpartanMC 18

Für die folgende Aufgabe soll nun eine Befehlsfolge ermittelt werden.

a = b + c;

d = e – f; a,b,c,d,e,f sind 18-Bit Worte im Speicher mit den Labels A,B,C,D,E,F

Der nachstehend angegebenen Rahmen soll dafür angewendet werden.

```
        .text       (0x050)
start:  lhi         r1, 0x210>>9        ;
        ...........                     ;  Laden Adresse A nach R1
        ...........                     ;
        ...........

stop:   j           stop                ; Endlosschleife

        .data       (0x210)
A:      .w18        0x.....
B:      .w18        0x.....
C:      .w18        0x.....
D:      .w18        0x.....
E:      .w18        0x.....
F:      .w18        0x.....
```

SpartanMC 18

```
        .text     (0x050)
start:  lhi       r1, A>>9          ;
        ori       r1, A&0x1ff       ;   Laden Adresse A = 0x210 nach r1
        l18       r2, 2(r1)         ; Laden b nach r2
        l18       r3, 4(r1)         ; Laden c nach r3
        add       r3, r2
        s18       0(r1), r3         ; a = b + c
        l18       r2, 8(r1)
        l18       r3,10(r1)
        sub       r2, r3
        s18       6(r1), r2
stop:   j         stop              ; Endlosschleife

        .data     (0x210)
A:      .w18      0x0               ; 3 + 5 = 8
B:      .w18      0x3
C:      .w18      0x5
D:      .w18      0x0               ; 9 – 6 = 3
E:      .w18      0x9
F:      .w18      0x6
```

SpartanMC 18

Zwei gleich große Datenfelder mit 18-Bit-Daten der Länge n sollen auf Gleichheit untersucht werden.

```
                .text       (0x50)
start:


                .data       (0x200)
adr_F1:         .w18                ; 1. Adresse von Feld1
                ..........
adr_F2:         .w18                ; 1. Adresse von Feld2

                .........
adr_n:          .byte               ; Länge der Felder F1 und F2
ergebnis:       .byte       0       ; Ergebnis: 0 für ≠ , 1 für =
```

SpartanMC 18

```
            .text       (0x050)
start:      lhi         r5, adr_F1>>9          ; die oberen 9 Bit von F1 in r5 laden.
            ori         r5, adr_F1&0x1ff       ; Adr. von F1 in r5
            lhi         r6, adr_F2>>9
            ori         r6, adr_F2&0x1ff       ; Adr. von F2 in r6
            lhi         r7, adr_n>>9
            ori         r7, adr_n&0x1ff
            l9          r7, 0(r7)              ; Länge der Felder in r7, Zähler für Test
            lhi         r8, ergebnis>>9
            ori         r8, ergebnis&0x1ff
            xor         r11, r11               ; clr r11
loop:       l18         r9, 0(r5)
            l18         r10, 0(r6)
            seq         r9, r10                ; cc = 1 wenn gleich, sonst cc = 0
            addi        r5, 0x1
            addi        r6, 0x1
            addi        r7, -1
            beqzc       store                  ; unterschiedliche Worte, Vergleichen beenden
            bnez        r7, loop
            addi        r11, 1                 ; alle gleich → r11 = 1
store:      s9          0(r8), r11
stop:       j           stop
```

SpartanMC 18

```
            .data       ()          ; Datenbereich

adr_F1:     .w18        1           ; 1. Adresse von Feld1
            .w18        2
            ..........
adr_F2:     .w18        1           ; 1. Adresse von Feld2
            .w18        2

            .........

lae         equ         (adr_F2 – adr_F1)/2

adr_n:      .byte       lae         ; Länge der Felder F1 und F1

ergebnis:   .byte       0           ; Ergebnis: 0 für ≠ , 1 für =
```

SpartanMC 18

; Berechnung von Y=X!, Y soll im Speicher an Adresse adr_y
; liegen, R5 enthält bereits X (es gelte X>1)

```
            .text       (0x50)
begin:      movi        r5, 9               ; Y = 9!
            lhi         r8, 0x100           ; Maske für Bit 17 in r5
            mov         r6, r5
            addi        r6, -1
loop:       mul         r5, r6
            movs2i      1, r4               ; SFR 1 = HIGH von MUL nach r4
            addi        r6, -1
            mov         r9, r5
            and         r9, r8
            bnez        r4, zugross
            bnez        r9, zugross         ; Vorzeichenbit ist gesetzt, Zahl würde Negativ
            bnez        r6, loop
mem:        lhi         r7, adr_y>>9
            ori         r7, adr_y&0x1FF
            s18         0(r7), r5
stop:       j           stop
zugross:    xor         r5, r5              ; 0 kann X! nicht werden.
            j           mem

            .data       (0x200)
adr_y:      .w18        0
```

SpartanMC 18

```
; Ausgabe einer Zeichenkette ab dem Symbol „text1“ bis zu einer Null

            .text       ()
begin:      LHI         R3,text1>>9
            ORI         R3,text1&0x1FF
loop:       L9          R1,0(R3)            ;Zeichen laden
            BEQZ        R1,ende_tx
            ADDI        R3, 1
            JALS        chout               ;Zeichen auf Konsole anzeigen
            J           loop
ende_tx:    TRAP        0

            .data       ()
text1:      .asciiz     „Hallo World\n“     ;Zeichen liegen im 9 Bit Abstand. Wird von
                                            ;text1 mit L18 geladen, steht im Rd 0x09061
                                            ;“H“ ist        0100 1000  |  0110 0001 ist “a“
                                            ;               00 1001 0000  0110 0001
```

SpartanMC 18

```
; 8 Bit Summe über eine Byte Liste bilden, Ende bei 0

            .text       ()
begin:      LHI         R7,liste1>>9
            ORI         R7,liste1&0x1FF
            XOR         R2, R2              ;Summe löschen
loop:       L9          R1,0(R7)            ;Byte laden
            ADDI        R7, 1               ;R7 := R7 + 1
            BEQZ        R1,ende_su
            ADD         R2,R1               ;Summe bilden
            SGTI        R2,255              ;CC = 1 wenn größer 255
            ANDI        R2,255              ;Nur 8 Bit übrig lassen
            IFADDUI     R2,1                ;8 Bit Übertrag addieren
            J           loop
ende_su:    J           ende_su             ;Summe in R2

            .data       ()
liste1:     .byte       1,2,3,4,5,6         ;Byte liegen im 9 Bit Abstand. Wird mit L18 von
            .byte       7,8,9,250,10,0      ;liste1 geladen steht im Rd 0x0202
```