Embedded Systems Architecture

Aufgabe 1: MIPS Assemblersprache

- Übersetzen Sie die folgende Funktion nach MIPS-Assembler.
- Beachten Sie dabei die MIPS-Register Konventionen.
- Pseudo-Instruktionen dürfen verwendet werden.

```
void clip(int a[], int n)
{
    int i;

    for (i=0; i<n; i++)
    {
        if (a[i]<0)
            a[i]=0;
        else if (a[i]>255)
            a[i] = 255;
    }
}
```

Aufgabe 1: Lösungsblatt

```
void clip(int a[], int n)
  {
   int i;

   for (i=0; i<n; i++)
       if (a[i]<0)
          a[i]=0;
       else if (a[i]>255)
          a[i] = 255;
}
```

```
clip:
   add $t0,$0,$0      # I = 0
   addi $t3,$0,255    # $t3 = 255
for:
   bge $t0,$a1,endfor   # if (i>=n) goto enfor
   sll $t1,$t0,2        # $t1 = 4*I
   add $t1,$t1,$a0      # $t1 = &a[i]
   lw  $t2,0($t1)       # $t2 = a[i]
if1:
   bge $t2,$0,if2       # if (a[i]>=0) goto if2
   sw  $0,0($t1)        # a[i] = 0
   j   endif
if2:
  ble  $t2,$t3,endif    # if (a[i]<=255) goto endif
  sw    $t3,0($t1)      # a[i] = 255
endif:
  addi  $t0,$t0,1       # i++
  j     for
endfor:
  jr    $ra             # return
```

Aufgabe 2: Zahlendarstellungen

- Stellen Sie die Zahl 19,75 in binärer Darstellung nach IEEE 754 mit einfacher Genauigkeit dar.
- Lösung:
  - s = 0
  - 19 = 2^4 + 2^1 + 2^0 = 10011
  - 0,75 = 2^-1 + 2^-2 = 0.11
  - 19.75 = 10011.11 = 1.001111 x 2^4
  - e = E + 127 = 4 + 127 = 131 = 128+3 = 2^7+2^1+2^0 = 1000 0011
  - f = .001111

0 1000 0011 0011 1100 0000 0000 0000 000

Aufgabe 3: Maschinensprache

- Stellen Sie folgenden Code in MIPS-Maschinencode (dezimal) dar:
  - Schleife beginnt an Adresse 1000
  - Verwenden Sie „MIPS Reference Card“
  - Beispiel Darstellung:

1000	1	2	3	4	5	6

```
L: sll   $t1,$t0,2
   add   $t1,$a0,$t1
   lw    $t1,-4($t1)
   beq   $t1,$a1,E
   addi  $t0,$t0,1
   j     L
E: …
```

Aufgabe 3: Maschinensprache Lösungsblatt

Embedded Systems Architecture

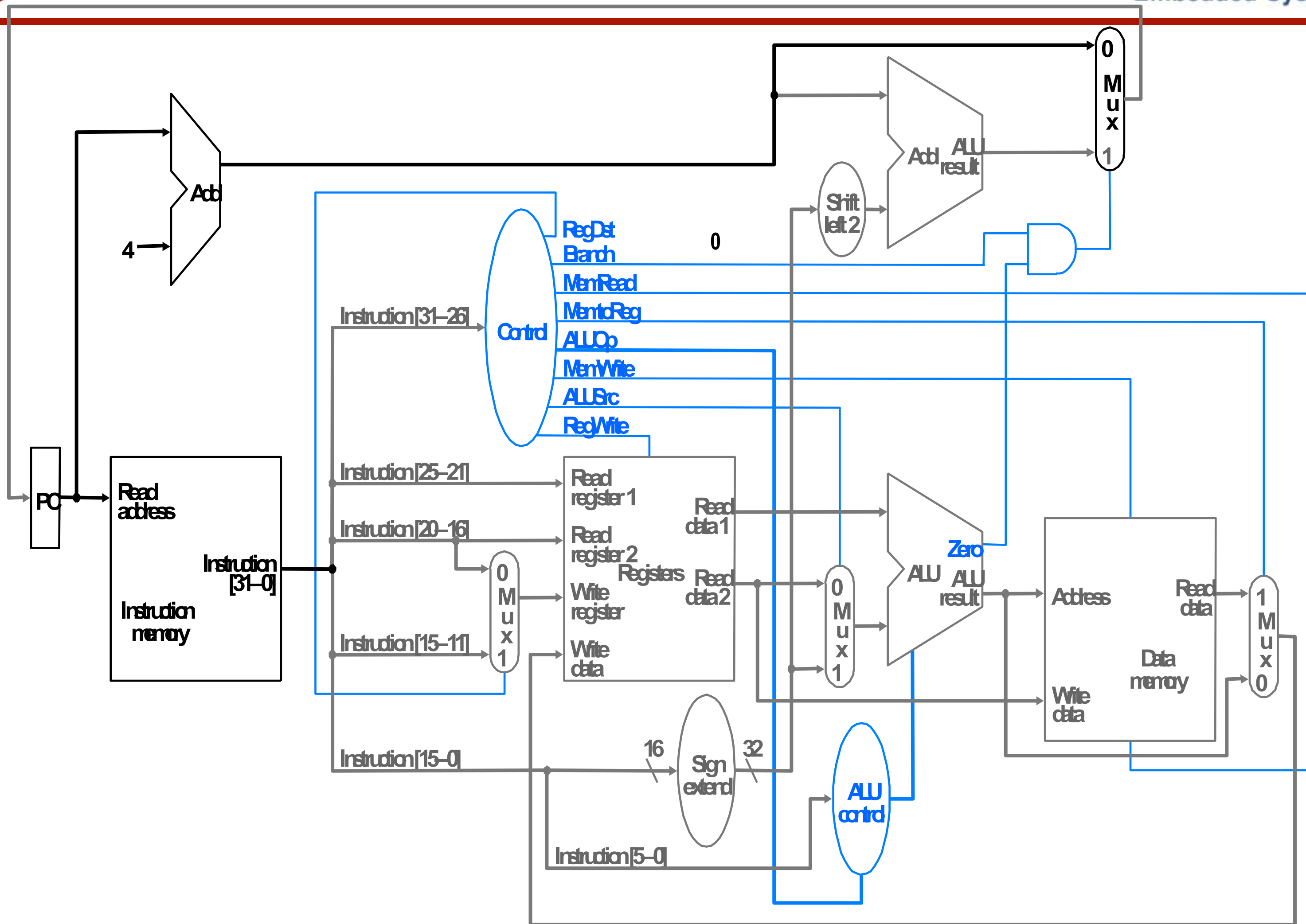
- Stellen folgenden Code in MIPS-Maschinencode (dezimal) dar:
  - Schleife beginnt an Adresse 1000

Adresse							Code
1000							L: sll $t1,$t0,2
1004							add $t1,$a0,$t1
1008							lw $t1,-4($t1)
1012							beq $t1,$a1,E
1016							addi $t0,$t0,1
1020							j L
1024	...						E: …

- Wir müssen den Eintaktprozessor um den Befehl `ori` (OR-immediate) erweitern.
  - Ergänzen Sie benötigte Datenpfade und Steuersignale in der Abbildung auf der nächsten Folie
  - Geben Sie die Werte an, die die Steuersignale haben müssen, so dass der Datenpfad den ori-Befehl ausführt. Verwenden Sie falls möglich Don’t Cares. Erweitern sie ggf. die ALU.
- `ori` Befehlsformat:

I-Format	opcode	rs	rt	imm
ori rt,rs,imm	0xd	rs	rt	imm

Aufgabe 4: Lösungsblatt (1)



Aufgabe 4: Lösungsblatt (2)

- Steuersignale:

RegDst	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegWrite

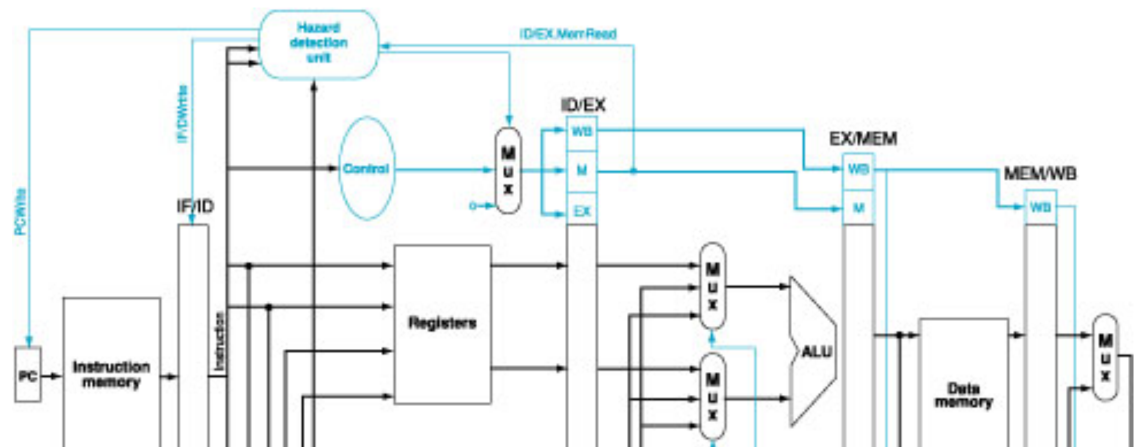
- *ALU Erweiterung:

ALUOp	Funct-Feld	ALU-Aktion	ALU-Steuereingang

Aufgabe 5: Pipelining

- Gegeben folgender Assemblercode
- Wird auf dem 5-Stufen-Pipeline-MIPS ausgeführt
- Vervollständigen Sie das Pipeline-Diagramm
  - Für jede Instruktion Kürzel IF, ID, EX, MEM, WB eintragen.
  - Wartezyklen(*stall cycles*) kennzeichnen mit X
  - Forwards kennzeichnen mit Pfeil (→) zwischen beteiligten Stufen

```
I0:  add $4,$1,$0
I1:  sub $9,$3,$4
I2:  add $5,$4,$7
I3:  lw  $2,100($3)
I4:  lw  $2,0($2)
I5:  and $2,$2,$1
I6:  beq $9,$1,target
```



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I0														
I1														
...														

Aufgabe 5: Pipelining - Lösungsblatt

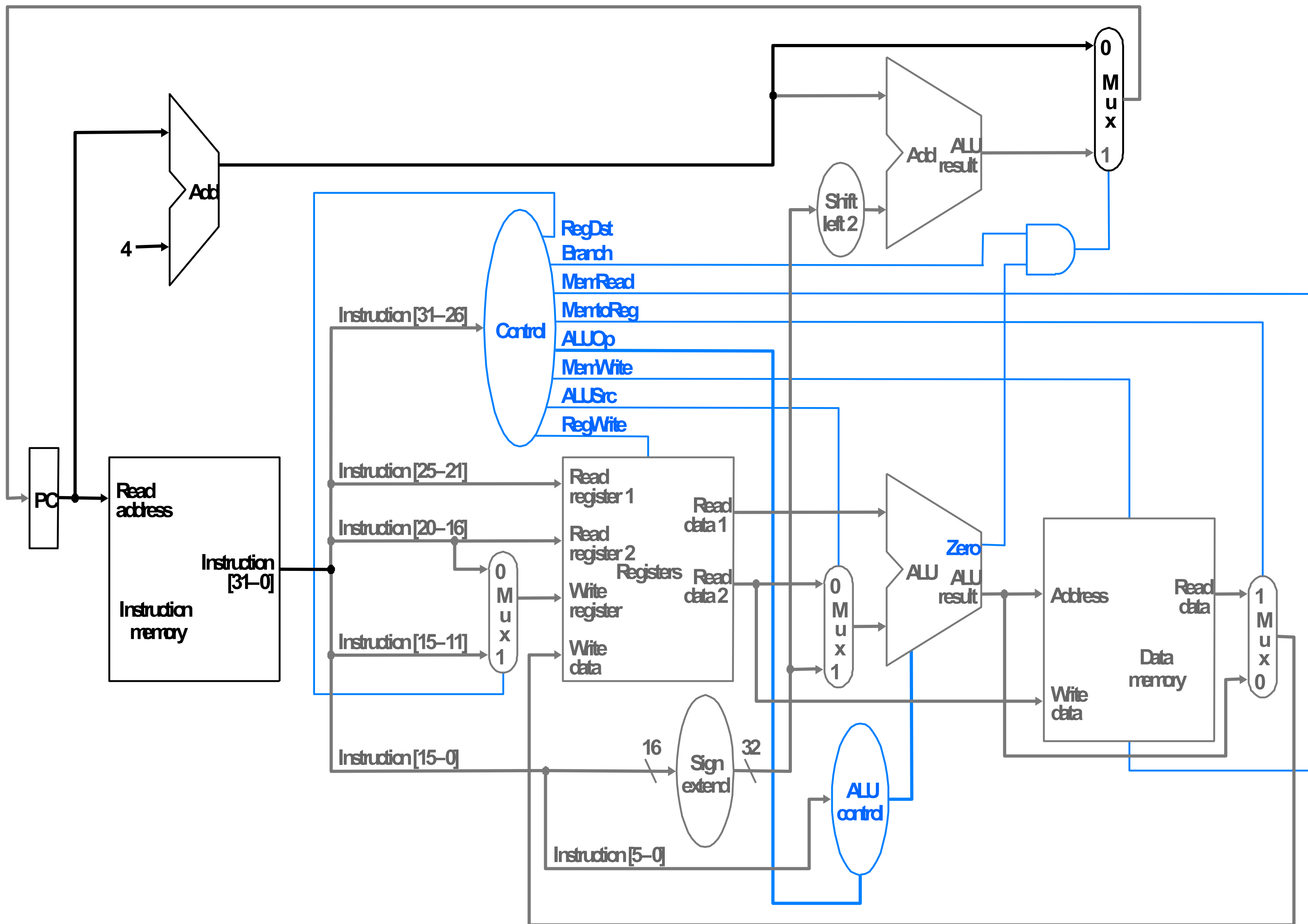

```
I0: add   $4,$1,$0
I1: sub   $9,$3,$4
I2: add   $5,$4,$7
I3: lw    $2,100($3)
I4: lw    $2,0($2)
I5: and   $2,$2,$1
I6: beq   $9,$1,target
I7: and   $9,$9,$1
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I0														
I1														
I2														
I3														
I4														
I5														
I6														
I7														

Aufgabe 6: Caches

- Gegeben ein Cache mit folgenden Daten:
  - 8-fach assoziativ
  - 64 Blöcke mit jeweils 32 B
  - Adresslänge: 32 Bit
- Wie viele Bits werden für den Index, Offset und Tag benötigt?

- Was ist ein „*Dirty Bit*“? Bei welcher Schreibstrategie wird es verwendet?



0
M
u
x
1
Add
ALU
result
Add
4
Shift
left 2
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
PC
Read
address
Instruction
[31–0]
Instruction
memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
Zero
ALU
ALU
result
Address
Read
data
Data
memory
Write
data
Instruction [15–0]
16
Sign
extend
32
ALU
control
Instruction [5–0]