Summer term 2023

Name, First name, Student ID

Examinant: Prof. Dr.-Ing. Rainer Keller	**Number of pages:** 10
Courses: Softwaretechnik & Medieninformatik Technische Informatik Ingenieurpädagogik	**Semester:** SWB4 TIB4 IEP4
Exam: **Computerarchitektur**	**Exam number:** IT 105 4003, SWB428
Aids: **none**, except calculator and **1 DIN A4** paper, both sides written (by **yourself)**	**Duration:** 90 minutes

Please read the instructions and questions carefully. Each task has sub-tasks, each of which is assigned a certain number of points. Each point is approximately one minute of time. Do use the available space and time and try to answer as elaborately, yet as thoroughly as possible. You may answer in English or in German.
Please note the **hints and instruction set** on the last pages.

General Questions (14 Points)

a) How many Electronic Control units (ECUs, "computers") does a modern car approximately have?

1

b) Name three common characteristics of microcontrollers:

1.	
2.	
3.	

3

c) Please name 3 upsides and 3 downsides of the C programming language?

Upsides:	
1.	
2.	
3.	
Downsides:	
1.	
2.	
3.	

5

Name, First Name, Student ID

d) In which situations should we use Assembler (ASM), in which rather high-level languages?

In which ASM?
In which high-level languages?

2

e) Why is endian-ness relevant when programming in a language like C? Please name two examples of functions:

Relevance endian-ness for high-level language:
Examples of functions:

3

Architectures (16 Points)

a) How is `int x=0xcafe` stored on HCS12, assuming x is located at `0x1022`?

Address	Old Value	Value
0x1020	0x01	
0x1021	0x00	
0x1022	0xff	
0x1023	0x0a	
0x1024	0x00	
0x1025	0x10	

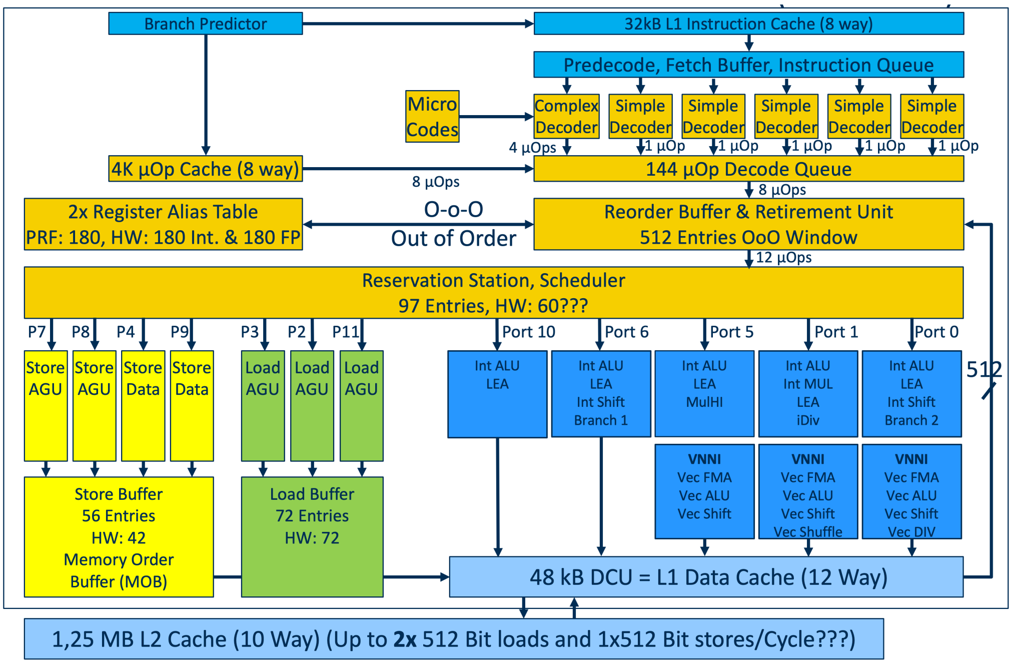
3

b) What endianness does Intel / AMD x86-64 provide?

1

Name, First Name, Student ID

c) Explain in your own words the note-worthy parts of current Intel Alder Lake P-cores – as shown in the architecture diagram below.



5

d) Is the x86-64 architecture **and** specifically the above Intel Alder Lake an Harvard- or a von-Neumann architecture? Please explain your argument?

Harvard- or von-Neumann architecture?
Please explain why:

4

e) Why is the Apple M2 SoC so performant?

3

Name, First Name, Student ID

HCS12 Assembler (29 Points)

a) How long is shortest & longest HCS12 instruction in binary encoding (in Bytes)?

Shortest:	
Longest:	2

b) Please state the error in the following instructions:

Instruction	Error
`LDAB #$cafe`	
`NEGD`	

2

c) Given the following code, state the value of registers D, X and Y after execution:

```
.const: SECTION
        ORG $C000
val1:   DC.B $57, $65, $6C, $6C, $20, $64, $6F, $6E, $65, $00
val2:   DC.W $12, $C00E
val3:   DC.L $89abcdef
.data:  SECTION
        ORG $2000
w:      DS.W 1
```

Instruction	D	X	Y
Initial values:	$cafe	$0123	$abcd
LDAB val1 LDX val2 LDY val3			
LDX #val1 LDD 1, X LDY #val2			

LEAY 2, Y
LDD 0, Y
LDX -2, Y			

LDX #$C
LDD [val1, X]
LDY 0, Y			

MOVW #$AABB, w
LDX #w
LDAA +1, X
LDAB 1, X+
TFR A, Y			

LDD #$1122 LDX #$3344 PSHD PSHX PULB PULA PULY			

6

Name, First Name, Student ID

d) State the value after executing the instructions and the CCR flags?

Instruction	**D**	**Negative (N)**	**Zero (Z)**	**Carry (C)**
Init: `LDD #$a55a`	`$a55a`	✓	-	-
`ROLA`				
`ANDB #$1`				
`ADDB #1`				
`MUL`				
`LSRD`				

5

The function `f` implements a std. LibC function with 1 parameter.
(Hint: write the results of each line on the right):

```
f:
    PSHX
    PSHY
    TFR D, Y
    LDX #0
L1:
    LDAB 1, Y+
    BEQ L2
    INX
    BRA L1
L2:
    TFR X, D
    PULY
    PULX
    RTS
```

e) What is the equivalent std. LibC function; Please state the signature?

4

f) What's the value in the `B` register when reaching Label L2?

2

____
Name, First Name, Student ID

g) The function f is inefficient – please optimize by rewriting f:

6

h) What does the instruction `BRA *+0` do?

2

Name, First Name, Student ID

C Programming (31 Points)

a) What is the content of variables S1 … S6 after execution of:

```
char a = 0x1, b = 0xa5, c = 0x10, S1, S2, S3, S4, S5, S6;
S1 = !b;
S2 = b & a;
S3 = b | c;
S4 = b ^ c;
S5 = ~b;
S6 = a >= b ? b : -c;
```

S1=	
S2=	
S3=	
S4=	
S5=	
S6=	

4

b) Consider You have the following function:

```
long f(char * a, int b, long c) {
    int d;
    …
}
```

Please state, where exactly the data is stored/passed, when using the HCS12 ABI?

	Stored where?	**How many bytes?**
Parameter a		
Parameter b		
Parameter c		
Variable d		
Return value		

4

c) When declaring a C function as Interrupt Service Routine (ISR), how many Bytes have to be saved on the Stack per each invocation of this Interrupt?

2

Name, First Name, Student ID

d) Fill in the following code to use the ADC to measure voltages. The analog input is in the range of 0 … 5V. First initialize the ADC using the ATD0 taking 8 samples using 10-bit resolution (right-adjusted) with the default 2MHz sampling clock from Channel 2. Convert the value back into a double-digit, fixed-comma value representing the range, i.e. 0 is 0V, 45 is 4,5V and 50 represents 5,0V etc.

```
void initADC(void) {
```


4

```
    ……
}

unsigned int getADC(void) {
```


4

```
}

unsigned int convert(unsigned int measured) {
```


1

```
}

void main(void) {
  initADC();
  while (1) {
      unsigned int measured = getADC();
      unsigned int converted = convert(measured);
      …
  }
}
```

e) What is the highest value `getADC()` may return?

1

____
Name, First Name, Student ID

f) Measure a signal on Port T.5 using the Input Capture mode for Channel 4, triggering on a both edges. Set the clock divider to 0. Store the current timer value to `lastTime`.

```
void initTimer(void) {
```

5

```
    ......
}
unsigned int lastTime;

void                          timer4Isr(void) {
```

3

```
}

void main(void) {
  initTimer();
  while (1) {
      … // uses lastTime
  }
}
```

g) Initialize the Serial Interface 0 to 57600 baud using 8 data bits, no parity and 1 stop Bit (8N1), and enable receiver and transmitter, but without interrupts.

```
void initSerial(void) {
```

3

```
}
```

Summer semester 2023

Name, First Name, Student ID

Hints:

Interrupt-Vector-Table

No.	Address	Usage
0	`$FFFE`	Reset
2	`$FFFA`	COP: Watchdog interrupt (**C**omputer **O**perating not **P**roperly?)
3	`$FFF8`	TRAP: Unimplemented opcode (Compiler bug? Stack clobbered?)
4	`$FFF6`	SWI: Software interrupt, called via instruction `SWI`
5	`$FFF4`	XIRQ: External, non-maskable interrupt request signal
6	`$FFF2`	IRQ: External interrupt request signal
7	`$FFF0`	**RTI: Real Time Interrupt**
8	`$FFEE`	**Timer Channel 0**
15	`$FFE0`	**Timer Channel 7**
20	`$FFD6`	SCI0: First serial interface
21	`$FFD4`	SCI1: Second serial interface
22	`$FFD2`	**ATD0**: Analog To Digital convertor
25	`$FFCC`	**PTH**: Digital input port H
127	...	...

RTI Interrupt:
Interrupt period $T_{RTI} = 2^{9+X} * (Y+1) / f_{OSCCLK}$. All values in ms @ f_{OSCCLK} = 4 MHz

	Y=0	Y=1	Y=2	Y=3	Y=4	Y=5	Y=6	Y=7	Y=8	Y=9	Y=10	Y=11	Y=12	Y=13	Y=14	Y=15
X=1:	0.256	0.512	0.768	1.024	1.280	1.536	1.792	2.048	2.304	2.560	2.816	3.072	3.324	3.584	3.840	4.096
X=2:	0.512	1.024	1.536	2.048	2.560	3.072	3.584	4.096	4.608	5.120	5.632	6.144	6.656	7.168	7.680	8.192
X=3:	1.024	2.048	3.072	4.096	5.120	6.144	7.168	8.192	9.216	10.240	11.264	12.288	13.312	14.336	15.360	16.384
X=4:	2.048	4.096	6.144	8.192	10.240	12.288	14.336	16.384	18.432	20.480	22.528	24.576	26.624	28.672	30.720	32.768
X=5:	4.096	8.192	12.288	16.384	20.480	24.576	28.672	32.768	36.864	40.960	45.056	49.152	53.248	57.344	51.440	65.536
X=6:	8.192	16.384	24.576	32.768	40.960	49.152	57.344	65.536	73.728	81.920	90.112	98.304	106.496	114.688	122.880	131.072
X=7:	16.384	32.768	49.152	65.356	81.920	98.304	114.68	131.072	147.456	163.840	180.224	196.608	212.992	229.376	245.760	262.144

ECT Timer: @ f_{BUSCLK} = 24 MHz
Clock period $T_{TCNT} = 1/f_{TCNT} = 2^X/f_{BUSCLK}$. In µs

x=0	x=1	x=2	x=3	x=4	x=5	x=6	x=7
0.042	0.083	0.167	0.333	0.667	1.333	2.667	5.333

Clock period $T_P = 2^{16} * T_{TCNT}$ Values in ms

x=0	x=1	x=2	x=3	x=4	x=5	x=6	x=7
2.731	5.461	10.293	21.845	43.691	87.381	174.763	349.525

PWM: @ f_{BUSCLK} = 24 MHz
Clock period of fast clock $T_{A/B} = 2^X/f_{BUSCLK}$. In µs

x=0	x=1	x=2	x=3	x=4	x=5	x=6	x=7
0.042	0.083	0.167	0.333	0.667	1.333	2.667	5.333

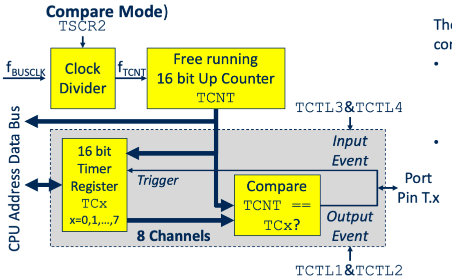
Clock period of slow clock $T_{SA/B} = 2*y*T_{A/B}$. In µs

	x=0	x=1	x=2	x=3	x=4	x=5	x=6	x=7
Y=1	0.083	0.167	0.333	0.667	1.333	2.667	5.333	10.667
…	…	…						
Y=255	21.250	42.500	85.000	170000	340000	680000	1360000	2720000

Serial Interface: @ f_{BUSCLK} = 24 MHz
Clock Divider Register SCIxBD $= f_{BUSCLK}/(16*f_{BIT})$

f_{BIT}=300 bit/s	600 bit/s	1,2 kbit/s	2,4kbit/s	4,8kbit/s	9,6kbit/s	19,2kbits	38,4kbit/s	57,6kbit/s	115,2kbit/s
5000	2500	1250	625	313	156	78	39	26	13

Every modern microcontroller has a powerful timer unit for tasks like
- Measuring time differences/timeouts in programs
- Measuring time instants of external events, pulse periods and pulse lengths of input signals (**Input Capture Mode**)
- Generation of interrupts and output signals at programmable times (**Output Compare Mode**)



The HCS12 timer unit consists of:
- A free-running 16bit counter TCNT with programmable clock frequency f_{TCNT}
- 8 channels (Port T), which can be used as inputs in "Input Capture Mode" or as outputs in "Output Capture Mode."

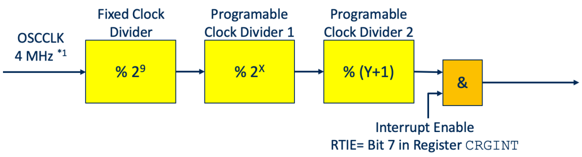
`TSCR1` (8 bit register)	**Enable the timer unit** Bit 7 = 1 — Set to 1 to **enable** Bit 6…0=0 — Default after reset (don't touch)
`TSCR2` (8 bit register)	**Set the timer clock frequency** Bit 7…3 = 0 — Default after reset (don't touch) Bit 2…0 — **Set for Clock divider x to 0**
`TIOS` (8 bit register)	**Select In- or Output Capture Mode per Channel** Bit y = 1 — Channel y in Output Capture Mode (y=0,1…,7) — Default y=0, i.e. Input Capture Mode
`TIE` (8 bit register)	**Interrupt Enable per Channel** Bit y = 1 — Channel y does generate interrupts y=0,1,…,7 — Default: y=0, i.e. no interrupt Each channel has its own ISR, which will be called by its associated input or output events.
`TFLG1` (8 bit register)	**Interrupt Flag indicates an timer interrupts even** Bit y = 1 — Channel y triggered an interrupt y=0,1,…,7 — Must be reset in the ISR by writing a 1 to the channel's bit in `TFLG1`.

Bit	7	6	5	4	3	2	1	0
`TCTL3` (8 bit)	Channel 7		Channel 6		Channel 5		Channel 4	
`TCTL4` (8 bit)	Channel 3		Channel 2		Channel 1		Channel 0	

Set the type of input trigger event:
- 00 Input not used
- 01 Positive edge
- 10 Negative edge
- 11 Positive & negative edge

`ATD0CTL2` (8 bit register)	**Enable the ADC and interrupts** Bit 7 = 1 — Enable the ADC module Bit 6 = 1 — Automatic resetting of the CCF flag Bit 5…2 = 0000_B — Miscellaneous options, don't change Bit 1 = 1 — Enable interrupt after conversion completes Bit 0 = 1 — Interrupt Flag, indicates an interrupt, must be reset by writing a 1 into this bit.
`ATD0CTL3` (8 bit register)	**Conversion sequence** Bit 7 = 0 — Default Bit 6…3 — Sequence Count, number of measurements Bit 2…0 = 000_B — Default
`ATD0CTL4` (8 bit register)	**Resolution and conversion speed** Bit 7 = 0 — 10 bit resolution (Bit 7=1 … 8 Bit) Bit 6,5 = 00_B — Sampling length 2 clocks (don't change) Bit 4…0 = 00101_B — Clock divider f_{ADC}=2MHz @ f_{BUSCLK} = 24 MHz (maximum clock frequency)
`ATD0CTL5` (8 bit register)	**Data format and start of conversion** Bit 7…5 = 100_B — Result in result register right-adjusted and unsigned Bit 4 — 0=single channel, 1=Multichannel conversion mode Bit 3 = 0 — Default Bit 2…0 — Channel select code, 1…7
`ATD0STAT0` (8 bit register)	Bit 7 = 1 — End of conversion EOC Bit 6…0 — Other status info, don't care
`ATD0DRx` (16 bit register)	Bit 15…0 — Result of 1st channel / the only channel in single channel mode. Where `x` is the number of the sample.

- Periodical interrupts can be generated using the RTI unit:



The interrupt frequency is defined via clock divider settings X=1..7 and Y:

Bit	7	6	5	4	3	2	1	0
Register `RTICTL`	0	X (3 bit)			Y (4 bit)			

$$f_{RTI} = \frac{f_{OSCCLK}}{2^{9+X} * (Y+1)}$$

Register `CRGINT`: To enable the RTI interrupt (RTI Enable, short RTIE), set bit 7 in `CRGINT` to 1. The **other bits** must not be changed!

Register `CRGFLG`: At the end of the ISR, the RTI Interrupt Flag (RTIF), i.e. bit 7 in `CRGFLG` must be reset by writing a 1. The other bits must not be changed!

The HCS12 of the Dragon12 has two UARTs SCI0 and SCI1. SCI0 is used for the debugger communication. SCI1 is free for user programs, e.g. to communicate with a terminal (Hyperterminal or terminal-component of the HCS12 debugger) running on a PC.

Each serial interface has 3 configuration registers, which must be configured once:

Baud Rate Register (x=0 for SCI0,…)	`SCIxBD` (16 bit register!)	Clock divider: `SCIxBD` $= \frac{f_{BUSCLK}}{16 * f_{bit}}$ (Dragon12 @ f_{BUSCLK} = 24 MHz)
Control Register 1	`SCIxCR1` (8 bit register)	Default after reset: 8N1 (8 data bits, no parity, 1 stop bit) If parity is required: Bit 0 = 1 — Uneven Parity (0 = odd Parity) Bit 1 = 1 — Use parity bit Bit 7..2 — Do not change
Control Register 2	`SCIxCR2`	Sender and receiver control Bit 2 = 1 — Receiver Enable Bit 3 = 1 — Transmitter (Sender) Enable Bit 5 = 1 — Receive Interrupt Enable Bit 7 = 1 — Transmit Interrupt Enable Set other bits to to Send and receive interrupts use the same interrupt vector. If both types of interrupts are enable, the ISR must poll status register SCIxSR1 to figure out, why the interrupt was triggered.

Sending and receiving uses the same data register:

Data Register	`SCIxDRL` (8 bit register)	If written: TX data register (for transmission) If read: RX data register (for reception)
	`SCIxDRH` (8 bit register)	Higher 8 bits of the data register. Used only if the SCI is configured for character length > 8 bit.

Polling the status of the serial interface:

Status Register 1	`SCIxSR1` (8 bit register)	Bit 7 = 1 — TX data register free Bit 5 = 1 — RX data register full = new character The other status bits indicate various error conditions, e.g. Bit 3 = 1 — Receive register overwritten by next character before the previous character was read. Bit 0 = 1 — Parity error
Status Register 2	`SCIxSR2` (8 bit register)	Optional for operating modes not described here.

Due to implementation details of the hardware, the status register SCIxSR1 should always be read, before reading or writing characters from/to the data register. This will automatically clear the status flags for polling and interrupt mode.