Embedded Controller

Vorlesung 6. Semester - Mechatronik

15.04.2004-8.07.2004

Inhaltsverzeichnis

1 Vorlesung vom 15.04.2004

1.1 Beispiele für ECS:

$klass.UR.:\quad T_{ur} := (1, 1, w) w >= 1$

$l' > 1:$ Makropipelining ($\rightarrow$ MISD)
Phasenpipelining: $w' > 1 \rightarrow$ Befehlsabarbeitung in w' Phasenunterteilung

$t_{i486} := (1, 1, 32 * 5)$
$t_{PentIII} := (1, \underline{1 * 5}_{(1)}, \underline{32 * 12 - 17}_{(2)})$
(1) pro Takt können 5 kodierte Befehle an Funktionseinheiten
(2) Phasenpipeline besteht aus 12...17 Schritte

$t_{ASCIRed} := (9000, 1, 32) \rightarrow 9000 PetniumPro$
$t_{CRAY-1} := (1, 1 * 122, 64 * 1 - 14)$
12 superskalare Funktionseinheiten
Pipeline bis 14 Stufen

1.2 Beispiele für Rechnerarchitekturen

PowerPC (IBM, Motorola): Performance Optimization with enhanced RISC
PC: Performance Chip

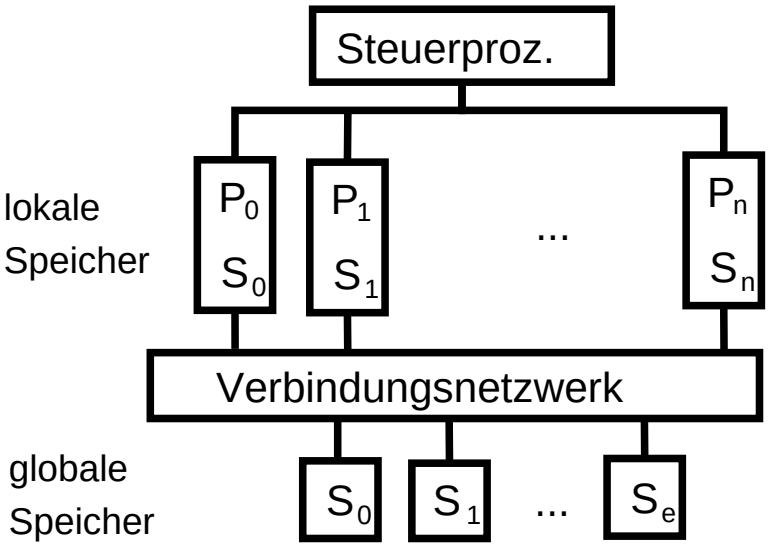
Sparc: Scalabel Processor Architecture

Parallelrechner Systeme

- SIMD
- MIMD
- speichergekoppelte MIMD-Rechner
- nachrichtengekoppelte MIMD-Rechner

2 Vorlesung vom 19.04.2004

2.1 Feldrechner mit lokalen/globalen Speichern



momentan leistungsfähigster Rechner SGI Origin 3800
128/64 Stck. MIPS R 12000
64 GB Hauptspeicher

2.2 heutige moderne Prozessoren nutzen:

- klass. Universalrechner-Architektur
- Phasenpipelining
- In-Order-Superskalarität
- Out-of-Order-Superskalarität
- Vektorprozessor-Architektur
- SIMD Nebenläufigkeit
- VUW-/EPIC-Architektur
- Simultaneous Multithreading
- Shared Memory Multiprozessor-Architecktur
- Distributed Memory Multiprozessor-Architecktur

3 Vorlesung vom 22.04.2004

3.1 Grundlegene Eigentschaften des C166

CPU:

- 4-stufige Pipeline
- interner ROM über 32-Bit-Bus verbunden → 2 Wortbefehle in einem Takt lesbar
- Registerbänke zu je 16 Register - Context-Switching in einem Takt möglich

Befehlssatz:

- meist verwendeten Befehl, 2 Byte, andere 4 Byte

RAM:

- Dual Port RAM, teilweise bitadressierbar
- External Bus Controler (EBC) → Zugriff auf ext. Speicher, multiplex/non-multiplex → Betriebsart/Buskonfiguration per Software
- Special Function Register (SFR, ESFR)
- Interrupt/PE-Controller

zusätzliche Pipeline-Effekte:

- laden de. Context Pointers: nicht sofort im nächsten Befehl auf neue Registerbank zugreifen → NOP
- laden des Datei Page Pointers: siehe oben
- explizites laden des Stack-Pointers

4 Vorlesung vom 26.04.2004

4.1 SYSCON-Register

16-Bit-Worte:

CLKEN:	TaktAusgabe an Port-PIN P3.15 (ANSI-C: P3^15)
BYTDIS:	Byte Disable → sperren von Byte-Zugriffen über EBC
ROMEN:	internes ROM freigeben/gesperrt
SGTDIS:	Segmentation Disable
ROMS1:	Mapping des internen ROM
STKSZ:	System Stack Size 000 : 256 Worte . . 111 : non-wrapping, beliebige Größe bis max 1024 Worte
BUSCON0..4	Buskonfiguration → "Ext. Buscontroller"
ADDRSEL1..4	Adressbelegung → "Ext. Buscontroller"

4.2 Programm Status Word (PSW)

Ergebnisse, arithmetische/log. Operationen

- N-negative
- C-carry
- Z-zero
- V-overflow

MULIP:

- Multipl/Divide in progress
- laufende Mult./Div. durch Interrupt unterbrochen

IEN:

- Freigabe/Sperre für alle indiv. Interrupts
  IEN=0: alle Int. gesperrt (bis auf Hardwareint. → Hardware TRAPS)!

ILVL:

- gerade aktuelles Interrupt-Priority-Level der CPU

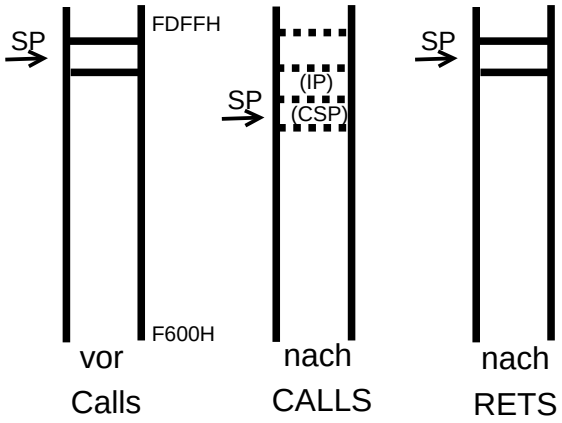
4.3 Speicheradressierung

wichtige Zeiger

- Code Segment Pointer
- Instruction Pointer
- Data Page Pointer DPP0...DPP3
- Context pointer → 16-Bit-Zeiger für aktuelle Registerbank (GPR)

R0	(CP)+0	F0H	R0..R7 als Byte-Register nutzbar
R1	(CP)+2	F1H	
.		.	
R15		FFH	

- Stack Pointer (SP) → zeigt Stack. Füllstand an
  - STKOV, STKUV



4.4 Special Function Register

- Trap Flag Register (TFR)
  Bsp:
  - STKUN - Trap Request Flag
  - STKOV - Trap Request Flag
  - ILLBUS - Zugriff auf ext. Bus wenn nicht verf.
  - ILLINA - Verzweigungsbefehl auf ungerade Referenzadresse
  - ILLOPC - ik. operand code
  - ILLOPA - ik. operand adress
  - UNDOPC - undefined opcode

  MDC: Mult/Divide Control
  MDH: Mult/Divide register high position
  MDL: Mult/Divide register low position

- ONES: const. one → anstelle FFFFH-const
- ZEROS: const. zero (Bit Befehle!)

4.5 Bitadressierbarer Speicher

- in SFR, ESFR, interner RAM je 128 Wort bitadressierbarer Speicher
- Adressierung über 8-Bit-Kurzadressen und Bit-Offset
  int. RAM: 0...7H
  Bsp.: RES.15 → Bit 15 aus RES
  Besonderheit: für SFR ESFR gelten gleiche Kurzadressen 80H..FFH, vor Nutzung der ESFR wird Befehl EXTR ausgeführt werden!

4.6 Beispiele für Befehle

Transfer-Befehle: MOV zielop,quellop

speziell:	SCXT (Switch Context)	- schreibt Inhalt von wordop1 auf Stack
	SCXT wordop1, wordop2	- schreibt wordop2 auf wordop1

5 Vorlesung vom 29.04.2004

5.1 Befehlssatz C167

Befehls-Operanden: GPR Wortreg.: R0 ... R15 → aktuelle Register

1. Register R_{wn}, R_{wm}
   R_{bn}, R_{bm} → Byte -Register Rl1,RH1 .. RH15
   SFR (WORTWEISE ADRESSIERT)
   reg

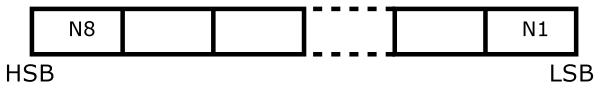
2. Speicher-Operanden:
   - direkt adressiert: 16-Bit-Wort (2 Bit. DPP, 14 Bit-Page-Offset)
   - indirekt
     $[R_{wn}]/[R_{wm}]$ R0..R15
     $[R_i]$ R0..R3

3. Konstanten: 3..16 Bit-Konstante
   #data16
   #data8
   #data4
   #data3

4. Bitadressen
   - badr → Adresse einer Bitspeicherstelle
   - beff → Adresse einer Befehlsspeicherstelle

5. Programmspeicher-Adresse
   - Caddr → 16-Bit-Adresse innerhalb 64-k-Segment
   - rel → Vorzeichen behaftete 8 Bit-Distanz zu Sprungziel
   - #trap7 - 7-bit-Interrupt-Nummern (0..127)

   Befehlsform: max 32 bit -> (4bit) Nibbles



   Bedingungscodes:
   CC_UC - unconditional
   CC_Z - CC-Zero(Z=1)
   CC_NZ - CC-NonZero (Z=0)

Multiplikation/Division

- MUL R_{wn}, R_{wm}
  Ergebnis in: MDH, MDL
- DIVLU R_{wn} Inhalt des Reg. Paares MDH/MDL
  Ergebnis: Quotient in MDL, Rest in MDH
  Z=1 bei Erg=0
  V=1 bei Division durch 0 und bei Ergebnis > 16bit

Unterprogramm-Aufruf

- CALL PC → Stack: Segmented Mode
  CSP → Stack
  IP → Stack
- Software-Interrupt: TRAP op1 (op1:int-Nummer)
  PSW → Stack
  CSP → Stack
  IP → Stack

5.2 Befehle für Einzelbit-Verarbeitung

Bit-Verarbeitung:

- Bit-Transfer,Set,Reset,Test, log. Verarbeitung
- BMOV Bitziel, Bitquelle
- MBSET Bitoperand
- BCLR op, BAND op1,op2
  (FLAG:Ergebnis → Z: NOR, V: OR, C: AND, N: XOR)

  Adressierung: spezielle Vordefinierte Symbole (IEN)
  Wortadresse.Bitstelle (PSW.11)
- Bit-Verzweigung JB op1,rel

5.3 Bit-Feld-Befehle

BFLDL Wortop, #andmask, #ormask (BitFieldLow: LowByte des Wort-Op wird gelesen
→ UND-Verknüpfung, mit #andmask
→ ODER-Verknüpfung mit #ormask

5.4 Programmierung in C

→ Controller !

- Daten sind "Bitmuster" (digitale Signale, Steuerbits, Messwerte)
- E/A über serielle/parallele Ports, analoge Schnittstellen
- Timer/Interruptsteuerung

ANSI-C + Erweiterungen

- Rotation
- Bit-Variable, Bitoperationen

Standard-Datentyp in "C":

- int vorzeichenbehaftete ganze Zahl (16bit)
- bei C167: unsigned int (Standard-Datentyp)

Bitvariablen

- sbit Bezeichner = bitoff^bitpos
  Bsp: sbit ein = P7^1
- bit Variablenliste
  Bsp: bit a=0, b=1

Bitoperatoren

- = Wertzuweisung
- ∼ Negation
- & UND
- | ODER
- ^ EODER

Bitfunktionen

- testclear_(bit)
- testclear_(bit)
- bfld_(bit)

6 Vorlesung vom 6.05.2004

6.1 Das Interrupt-System des C167

- Non maskable Int. (NMI)
  → ausgelöst durch negat. Flanke am PIN NMI
- Hardware Traps
  → nicht maskierbare Int. mit fester Priorität (Bsp: STCKOV - StackOverflow)
- Hardware Int. der integrierten Peripherie
  → verschiedene Level, einzeln o. gesamt sperrbar
- Hardware Int. der externen Peripherie
  → verschiedene Level, einzeln o. gesamt sperrbar
- Software Traps TRAP int*

Prioritätsstruktur

1. Request Priority
2. Service Priority - NMI, STACK-Traps

speziell -Interrupt-Quellen

- 32 Int. der CAPCON-Einheiten (auch als ext. Interrupts)
- 9 Int. Timer Einheiten (3 als ext. Int.)
- 2 Int. der A/D-Wandler
- 6 Int. der zwei seriellen Schnittstellen
- 1 Int. der Pulsweiten-Modulations-Einheit

Beispiel

- Fast External Interrupt am PIN 2-9 (Eingang EX1IN); positivie Flanke
- Interrupt-Behandlung: Zustand des Bit0 an Port2

```
NAME main
EXTRN init_HW_int:NEAR
...
stack definieren
...
lock segment initialisieren
...
MOV SP,#top_of_stack
OR DP2,#0FF ; Direction Port2=Output
```

Initialisierung:

```
                NAME hw_int
                PUBLIC init_hw_int
        ....
        Init codesegment
        ...
Hardware-Interrupt Routine
ist_hw_int          PROC INTERRUPT hw_isr=19h        // Sprungbefehl zur Int.Routine
                                                     // in Int.-Tabelle
                        XOR P2,#01h
                        RETI
ist_hw_int          ENDP
```

7 Vorlesung vom 13.05.2004

Änderung: bei jedem Int. Bit 0 an Port 3 invertiert

EXICON:

00	00	01	00

EXI1ES 01-Int. bei positiver Flanke

CC91C: Int.-Controll-Register

	JR	JE	JCVL	GL
..	0	1	0001	00

```
...
PROC
MOV SP,#top_of_stack
OR DP3,#0FFh -> Port3 -> Output
CALL init_HW_Int
MOV P3,#0h
endless JMP endless
ENDD
.
.
HW-Interrupt-Behandlugns-Routinge
isv_hw_int PROC Interrupt hw_isr=19h
XOR P3,#01h -> Bit0 von Port3 invertieren
RETI
isv_hw_int ENDP
.
.
Initialisierung
PROC
EXTR#1 ,ESFR ansprechen
MOV EXICON, #000 0100b
MOV CC91C, #0100 0100b
BSET IEN
RET
ENDP
```

PEC-Interrupt int. am P2.9 (Eingang EX1IN)

- initialisieren auf PEC0
- bei jedem INT Wert aus Tabelle an Port3
- nach 10 Werten soll Int-Routine zur Vorinitialisierung aufgerufen werden

EXICON

00	00	01	00
		EX	1ES

01-Int. bei pos. Flanke

CC91C

...	0	1	1110	00
	JR	JE	JCVL	GL

GL→Gruppenlevel

PECC0

....	10	1	0000	1010
	INK	BWT	COUNT	

BWT→Byte oder Wort-Transfer

```
main PROC
        MOV SP,#top_of_stack
        MOV  DP3,#0ffh, Port 3 Output
        CALL init_HW_PEC
        MOV P3,#00h
endless JMP endless
        END
```

INT-Routine

```
        isr_hw_pec PUBLIC int_HW-PEC
                PROC INTERRUPT pec_isr=19h
                MOV R4,#tab
                MOV SRCR0,R4
                MOV R4,P3
                MOV DSTP0,R4
                MOV PFCC0,#101000010100
                RFTI
        isr_hw_pec ENDP
```

Initialisierung

```
init_hw_pec PROC
        EXTR #1
        MOV Exicon,#0000100b
        MOV CC91C, #01111000b
        MOV R4,3tab
        MOV SRCP0,R4
        MOV R4,P3
        MOV DSTP0,R4
        MOV PECC0#10100001010
        BSET IEN
        RET
init_hwpec  ENDP
        ...
        tab DB 1,2,3,4,5,6,7,8,9,10
```

PEC=Transfer mit Interrupt an P7.7
Aufg.:

- jede fallende Flanke an P7.7 überträgt 1 Wort aus Tabelle
- von 10 Elemente an PORT 2
- nach 10 Worten → INT-Neuinitialisierung → Übertragung beginnt von vor → Abbruch mit Taste an P7.0

```
#include <reg167.h>
sbit lauf = P7^0;
unisgned int tab[10];

int main (void) {
        unsigned int i;
        for (i=0; i< 10 ; i++) tab[i]=i+1;
        DPR=oxffff; // Port 2 Ausgang
        PR=0;
        PECC0=0x40a; //
        DSTP0=(unsigned int) &P2; // Zielzeiger
        SRCP0=(unsinged int) &tab;  // Quellzeiger
        CCM7=Com7|0x2000h; // fallende Flanke an P7.7
        CC31IC=0x78h; //011111000

        while (lauf);

        IEN=0;
        return 0;
  }

void neu (void) interrupt 0x46h { //P7.7 Int-Programm
        PECC0=0x40h;
        DSTP0= (unsigned int ) &P2; // Zielzeiger
        SRCP0= (unsigned int) &tab; // Quellzeiger
}
```

8 Vorlesung vom 27.05.2004

Beispiel: T3 als Frequenzteiler; am Ausgang P3.3 erscheint Rechteckfrequenz von 1,2Hz Prozessortakt steuert

```
/* Timer T3 als Frequenzteiler 20 MHz ->1,2 Hz */
#include <reg167>

sbit lauf = P7^0;
sbit richt = DP3^0;
sbit daten = P3^3;

int main (void) {
        richt = 1;  // P33 ist Ausgang
        daten = 1; // Daten high
        T3CON=0x245;

        while(lauf); // solange lauf high
        T3R=0;  // Ruecksetzen des Timers
        return 0;
}
```

Beispiel: bei fallender Flanke an T4N (P3.5) wird aktueller Inhalt von T3 gespeichert nach T4, auf Port2 ausgegeben und T3 d.h. Zeitmessung zwischen Programmstart und Eintreffen des Triggersignals, Überlauf des Zählers erzeugt Blinken am Ausgang T30

```
inc. ...
(sbit s.o.)
int main(void) {
        DP2=0x0fff; //Port2 -> Ausgabe
        P2=0;
        richt=1;
        daten=1;

        T3CON=0x207; //000001000000 1111
        T3=0;
        T4CON=0x2a;

        T4IC=0x44 // T4IE=1, ILVL=1, GLVL=0
        T3R=1; // T3 ein
        IEN=1;  //alle INT erlaubt

        while (lauf);

        IEN=0; // alle INT gesperrt
        T3R=0; // T3 gesperrt
        return 0;
}
```

```
void aus(void) interrupt 0x24 {
        P2 = T4; // Port2 <- Capture-Wert
        T3 = 0;
}
```

Beispiel 3: Compare betrieb: Ausgabe zweier phasenverschobener Rechtecksignale T8 läuft mit 2,5 MHz wird 65536 Takten entsprechend 26 ms (38 Hz) nachgeladen CC31 mit konstantem Vergleichswert - Schaltwert P7.7 alle 26 ms um und gibt Rechtecksignal von 19 Hz aus CC30 gibt ebenfalls 19Hz aus aber phasenverschoben, erreicht durch variablen Vergleichswert an P2 (bei 8000H in der Mitte der Timerperiode)

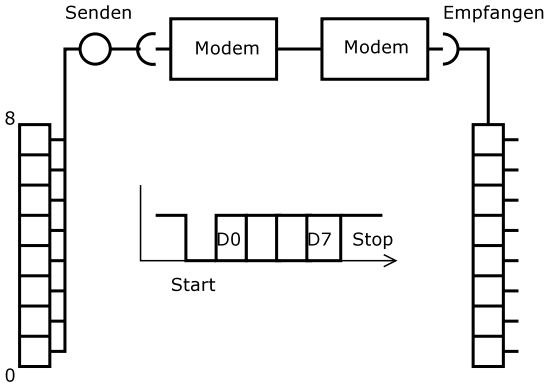
```
(incl ...)

sbit lauf=P7^0;
sbit DP77=DP7^7;
sbit DP76=DP7^6;

int main(void) {
        DP77=1;
        DP76=1;
        CCM7=0xdd00;
        T78CON=0x400;
        CC31=0;
        while (lauf) {
                CC30=P2;
        }
        T78CON=0; // T8 sperren
        CCM7=0;  // CC-Kanaele sperren

        return 0;
}
```

9 Vorlesung vom 10.06.2004

Beispiel: auf Kanal Nr. 3 (P7.3); 20 MHz 65536 = 305 Hz, Tastverhältnis einstellbar, low-Zeit in PW3 von Port2 Steuerbit P7.3 von Port 7.4 übernommen, für P7.3=1 wird Ausgangssignal invertiert

```
#include

sbit lauf=P7^0;
sbit DP73=DP7^3;
sbit P73=P7^3;
sbit DP74=P7^4;

int main(void) {
        DP73=1;
        P73=1;
        PP3=0xffff; // max periode

        PW3=0x8000; // Anfangswert

        PWMCON0=0x0008;
        PWMCON1=0x0008;

        while(lauf) {
                PW3=P2; // Low-Zeit von Port2
                P73=P74; // Datenbit von P7.4
        }

        PWMCON0=0; // Timer aus
        PWMCON1=0; // Kanal 3 aus

        return 0;

}
```

9.1 serielle Kommunikation



$$SOBRL = \frac{Systemtakt}{16 + (2 + SOBRS) + Bandbreite} - 1$$

Beispiel: Serielle Schnittstelle 9600 baud, 8 Datenbit, ein Stopbit, ohne Parität kein Interrupt, sondern Software-Abfrage → Gegenstation ist PC, Terminal Programm, das gesendete Zeichen auf PC-Bildschirm darstellt Abbruch mit ESC (1Bit)

```
incl.
sbit dp310 = DP3^10; // Richtung Ausgang (TxD)
sbit p310 = P3^10;  // Portbit TxD
sbit dp311=DP3^11;  // Richtung Eingang (RxD)


//UP wartet auf Zeichen
unsigned int empf(void) {
        while (!S0RIR);
        S0RIR=0;
        retrun S0RBUF; // Zeichen abholen
}

//UP sendet Zeichen seriell
void send (unsingend int zeichen) {
        while(!S0TIR);  // warte bis Sender frei
        S0TIR=0;
        S0TBUF=Zeichen; // Zeichen nach Sender
}

int main(void) {
        unsigned int x; // Hilfsvariable

        dp310=1; // TxD
        p310=1; // TxD=high(Ruhe!)
        dp311=0; // RxD
        S0TIC=0x0080; // Sende frei
        S0RIC=0x0000; // Empfaenger hier
        S0BG=64; // 9600 baud
        S0CON=0x8011; //asyncron, 8bit, 1stop


// neue Zeile und prompt
        send(10); //LF
        send(13); //CR
        send('>'); // Prompt>

        // Schleife bis ESC
        do {
               x=empf(); // Zeichen von Konsole
                send(x); // Ausgabe
```

```
        } while (x!=0x1b); // solange kein ESC

}
```

10 Vorlesung vom 17.06.2004

Beispiel A/D-Wandler:

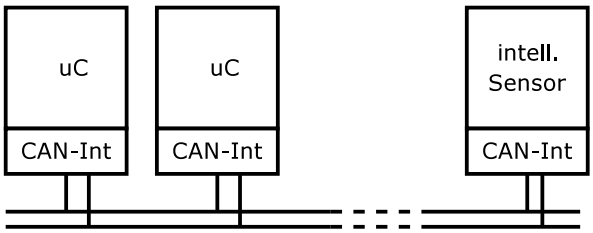
- einmalige Wandlung des Kanals AN0 (P5.0) bei fallender Flanke an P7.7
- gewandelter-Wert wird ausgegeben an Port2
- Ende der Wandlung durch Kontrolle des Anzeigeflags ADCIR

```
#include <reg167.h>
sbit lauf = P7^0;
sbit warte = P7^7; // startet Wandlung

int main (void) {
    P2=0; // Ausgabe loeschen
    DP2=0xffff; // setze Port auf Ausg
    while(1) {
        while(warte && lauf) ; // warte auf fallende Flanke P7

        if (!lauf) break;  // Abbruch bei P7.0 low

        ADCON = 0xf080 // 1111 0000 1000 0000 -> Start

        while (!ADCIR);  // warte bis Ende der Wandlung

        ADCIR=0;
        P2=(ADDAT & 0x03ff) >> 2;

        while(!warte); // steigende Flanke an P7.7
    }
    P2=0;

    return 0;
}
```

10.1 Anschluss externer Peripheriebausteine

- Adressierung + Erzeugung erforderlicher Steuersignale (RD, WR, CS) incl. deren Timing durch Steuerregister
  BUSCON0..4
  ADDRSEL0..3

10.2 CAN - Controller Area Network



- verteilte Systeme (Multi Master) unterstützt
- Übertragungsraten bis 1 Mbps
- rel. hohe Störsicherheit
- gesichert Datenübertragung durch CRC

Aufbau einer Nachricht:

SOF	Identifier	RTR	Control	Datenbytes	CRC	ACK	EOF
1	11	1	6	0..8x8	15	2	7
1	29	1	6	0..8x8	15	2	7
full CAN							

SOF: Start of Field
RTR: RemoteTransmission Reg
dominant ("0") bei Datenframe
rezessiv ("1") bei Remoteframe
Control: Anzahl der gesendeten Datenbyte

Antwort auf Anfrage (Datenframe) setzt sich durch gegenüber Anfrage

11 Vorlesung vom 24.06.2004

11.1 Programmstruktur eines Assembler-Programms

Modul:

- Übersetzungseinheit → Name (Linker/Locater)

```
NAME  modulname
...
END
```

Section:

- zusammenhängendes Stück Speicher = Segment die x86-Familie
- segm. /nonsegm. Mode
- segmented Mode: ASSUME-Befehl → laden der Segment-Register

```
section_name SECTION type [align][combine]['class.name']

section_name ENDS
```

type: CODE, DATA, BIT

align: BIT, BYTE, WORD

combine: Anordnung der Segmente
PUBLIC .. Linker legt fest
AT adresse .. feste Adresse
SYSSTACK

Procedure-Deklaration:

Normale Prozedur

```
procname PROC proctype
       ... code
       RET
procname ENDP
```

proctype: NEAR innerhalb 64kB
FAR nicht innerhalb 64kB

Interrupt Prozedur

```
procname PROC INTERRUPT intname[=intnumber] USING regbank
    ... code
    RET
procname ENDP
```

→ Assembler generiert für Interrupt-Vektor-Tabelle einen Sprungbefehl zur Interrupt-Routine

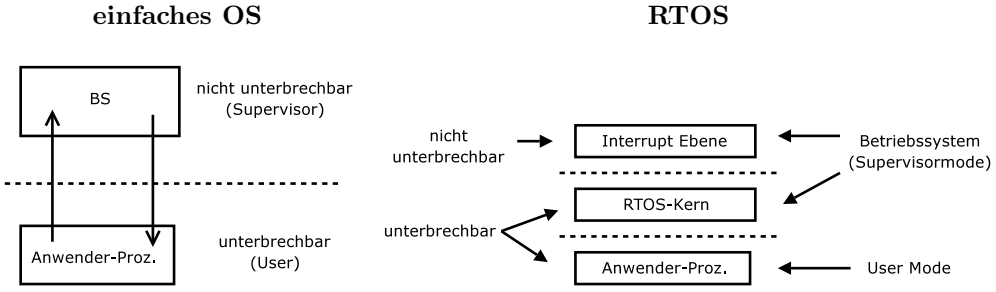
intname: sybmolischer Name

intnumber: feste Int.-Nr.

11.2 Echtzeitbetriebssysteme (Real Time Operating System)

- "Infrastruktur" zur Verwaltung eines Prozess-Steuer-Systems (Embedded Systems)
- Aufgaben, die unabhängig von konkreten Prozess, bei jedem Einsatz nötig → sinnvollerweise zusammengefasst → Real-Time-OS

Aufgabe eines RTOS: Rechenprozessen zu jedem Zeitpunkt die erforderlichen Ressourcen "just-in-time" zur Verfügung zu stellen



Anforderungen an RTOS:

- garantiert minimale Reaktionszeit
- dynamische Prioritätssteuerung der Tasks
- Task-Synchronisation und -Kommunikation
- Rom-fähig, skalierbar in Größe
- Zwei Schnittstellen
  - Bedienoberfläche zur Bed.+Steuerung des BS
  - Programmierschnittstelle für Systemcalls

11.2.1 Task-Modell

- Task: dynamische Ausführung des Programmmodules
- Task durch BS verwaltet

11.3 Echtzeit-Scheduling-Algorithmen

Bsp:

1. Rate Monotone Scheduling(statisch) → RMS

Prozess A:	alle 30ms	→	33 mal/s	→	Priorität 33
B:	alle 40ms	→	25 mal/s	→	25
C:	alle 50ms	→	20 mal/s	→	20

2. Earliest Deadline First → EDF

Wann ist ein System "schedulbar" ?

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

im Bsp.:

$A : \frac{10}{30}$ $B : \frac{15}{40}$ .. sum=0,808 $C : \frac{5}{50}$

RMS funktioniert nur bei nicht zu hoher CPU-Last

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq m(m^{\frac{1}{m}} - 1)$$

12 Vorlesung vom 1.07.2004

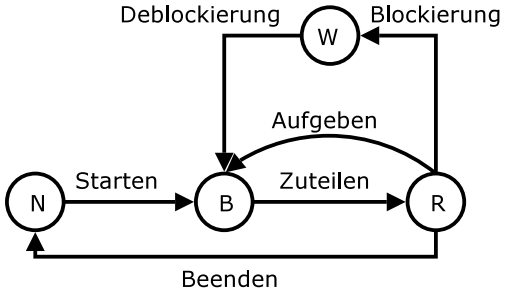
Zustände in denen sich ein Task befinden kann:

N nicht aktiv → Programm nicht gestartet

B bereit → ein Task wartet auf Zuteilung der CPU durch Scheduler

R rechnend → die Task, die CPU besitzt (bei Single Prozessor System)

W wartend → eine oder mehrere Tasks, die auf des Eintreten eines Ereignisses warten



12.1 Elemente des OSEK-Betriebssystems

Taskverwaltung zwei Tasktypen:

1. Basic Tasks:
   für kurze abgeschlossene Funktionalitäten mit definierter Laufzeit.
   3 Zustände: N, R, B
2. Extended Tasks:
   können auf asynchrone Ereignisse warten, ohne Rechenzeiten zu belegen (OS kann Rechenzeit andere Tasks abarbeiten
   4 Zustände: N, R, B, W

Eventsteuerung ermöglicht dem Extended Task Synchronisierung auf andere Tasks oder externe Ereignisse

Counter und Alarme Verwendung für zeitabhängige Steuerungsaufgaben

Interruptverwaltung Nutzung von Betriebssystemfunktionen (oder nicht)

Ressourcenverwaltung Taskpriorität zur Laufzeit dyn. verändert so, dass konkurrierende Tasks nicht gleichzeitig auf Ressourcen zugreifen

Fehlerbehandlung Mechanismen für zentrale als auch dezentrale Fehlerbehandlung

13 Vorlesung vom 8.07.2004

13.1 Prüfungsschwerpunkte

- grundsätzliche Dinge bzgl. CISC, RISC, Pipelining, Superscalarität
- wann Microprozessor , wann Microcontroller (Gemeinsamkeiten, Unterschiede)
- grundsätzlicher Aufbau C166, welche Teile, welche Taktfreq, wie viel Bit was für Peripherie-Einheiten verfügbar
- Interrupts - wie funktionieren, welche Voraussetzungen, Programmierung, Stufen
- PEC Transfer - entsprechend Interrupt, was passiert
- grundsätzliche Funktion der Peripherie
- Adressierungsmodelle (Hauptspeicher → Programm, Daten) → vorgegebenes Speichermodell Adressen berechnen etc.