

Prozesse vs. Threads

- Verwaltungsaufwand von Threads ist deutlich geringer
- Effizienzvorteil: bei einem Wechsel von Threads im gleichen Prozessraum ist kein vollständiger Austausch des Prozesskontextes notwendig.
- Kommunikation zwischen Threads des gleichen Prozesses kann über gemeinsamen Speicher erfolgen.
- Zugriffe auf den Speicherbereich anderer Prozesse führen zu Fehlern.
- Probleme bei Threads: durch die gemeinsame Nutzung von Daten kann es zu Konflikten kommen.



Nebenläufigkeit

Unterbrechungen

Binding Rechnersystem-Umwelt

- Es muss ein Mechanismus gefunden werden, der es erlaubt, Änderungen der Umgebung (z.B. Druck einer Taste) zu registrieren.
- **1. Ansatz:** Abfrage (Polling)
Es werden die E/A-Register reihum nach Änderungen abgefragt und bei Änderungen spezielle Antwortprogramme ausgeführt.
 - Vorteile:
 - bei wenigen EA-Registern sehr kurze Latenzzeiten
 - bei einer unerwarteten Ereignisflut wird das Zeitverhalten des Programms nicht übermäßig beeinflusst
 - Kommunikation erfolgt synchron mit der Programmausführung
 - Nachteile:
 - die meisten Anfragen sind unnötig
 - hohe Prozessorbelastung
 - Reaktionszeit steigt mit der Anzahl an Ereignisquellen

Lösung: Einführung des Begriffs der Unterbrechung

- **2. Ansatz:** Unterbrechung (Interrupt)
- Eine Unterbrechung stoppt die Verarbeitung des laufenden Programms. Die Wichtigkeit des Ereignisses, welches die Unterbrechung ausgelöst hat, wird überprüft. Darauf basierend erfolgt die Entscheidung, welche Reaktion erfolgt.
- Vorteile:
 - Prozessorressourcen werden nur dann beansprucht, wenn es nötig ist
- Nachteile:
 - Nicht-Determinismus: Unterbrechungen asynchron zum Programmablauf (und zum Prozessorzustand) eintreffen.

Unterbrechungen

- **Unterbrechungen:** Stopp des Hauptprogrammablaufs, Aufnahme der Programmausführung eines „Unterbrechungsbehandlers (UBB)“ an einer anderen Stelle; nach Beendigung des UBB (zumeist) Rückkehr an die Stelle des Auftritts der Unterbrechung im Hauptprogramm.
- **Synchrone** Unterbrechungen: treten, falls sie auftreten, immer an *derselben Stelle* im Programmcode auf. Man bezeichnet sie auch als *Traps* oder *Exceptions* bzw. „Software-Interrupts“
- **Asynchrone** Unterbrechungen: Auftrittszeitpunkt ist unbestimmt; es kann nicht gesagt werden, an welcher Stelle der Hauptprogrammausführung der Prozessor zum Zeitpunkt der Unterbrechung ist. Asynchrone Unterbrechungen werden auch als Interrupts bezeichnet; weil sie von der Hardware-Peripherie erzeugt werden, auch als *Hardware-Interrupts*. Sie üben „Brückenfunktion“ zwischen Hardware und Software aus.

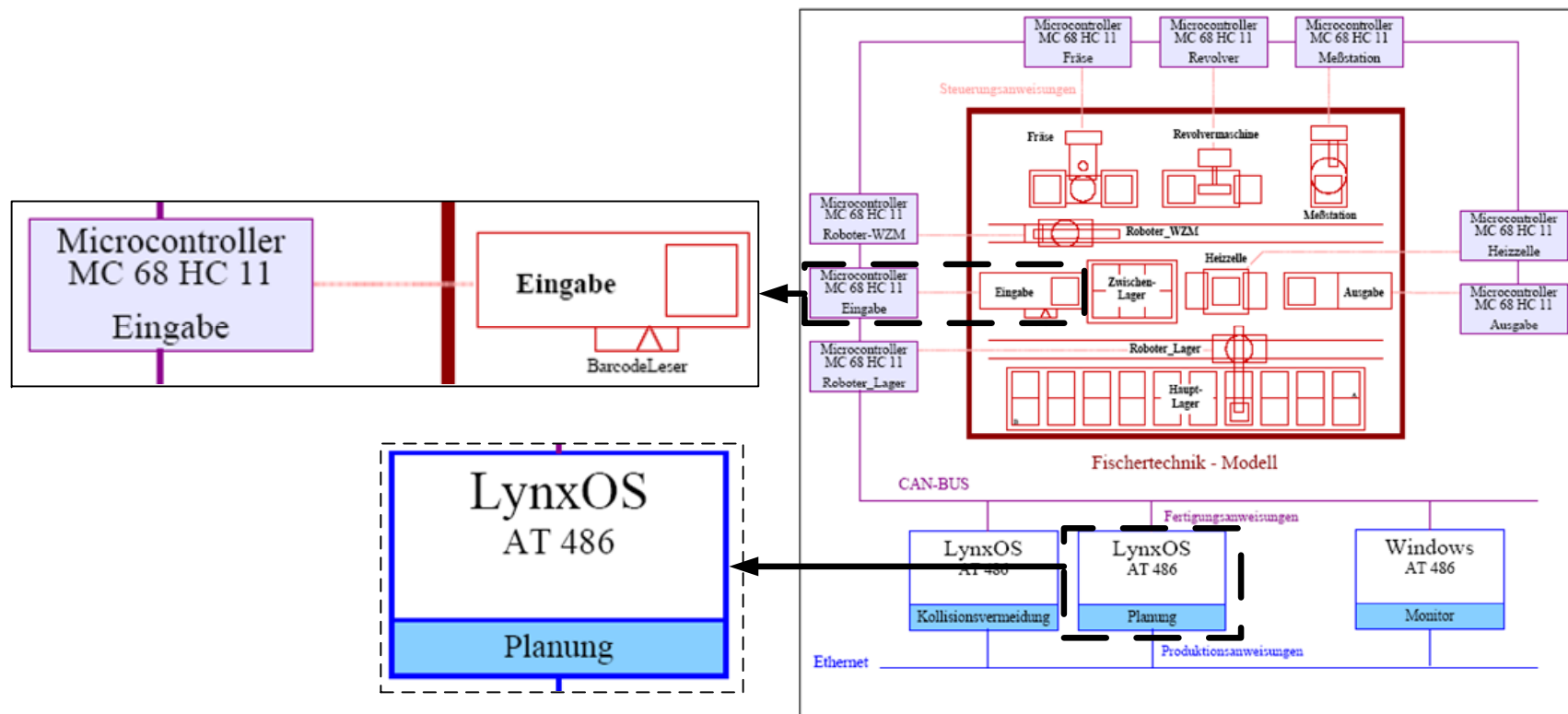
Synchrone Unterbrechungen (Traps/Exceptions)

- Werden durch das Programm selbst ausgelöst, d.h. dasselbe Programm, ausgeführt mit denselben Parametern wird in der Regel an derselben Stelle dieselbe Unterbrechung auslösen (vorhersagbar in dieselbe „Falle“ laufen)
- **Auslösung bei Fehler** – Ausnahme/Exception, Beispiele:
 - Arithmetikfehler (Division by zero, overflow, not-a-number NaN, ...)
 - Speicherfehler (Page Fault, segment Fault, memory full, ...)
 - Befehlsfehler (Illegal instruction, privileged instruction, bus error, ...)
 - Peripheriefehler (End-of-file EOF, channel blocked, unknown device, ...)
- Bei Exceptions **nur dann** Rückkehr an den Auftrittspunkt, wenn die Fehlerbedingung im Ausnahmebehandler beseitigt werden kann, andernfalls Abbruch (resumption vs. termination)
- **Auslösung durch spezifische Instruktion**: Breakpoint, SWI, TRAP, INT, ... entweder zum Zwecke des „Debuggings“ oder zum Aufruf von Betriebssystem-Diensten (z.B. MS-DOS „INT 21h“, siehe z.B. http://en.wikipedia.org/wiki/MS-DOS_API)
- Traps können auch benutzt werden, um einen Hardware-Interrupthandler zu testen.

Asynchrone Unterbrechungen (Interrupts)

- Werden durch externe Prozesse ausgelöst, d.h. sind bezüglich des genau-en Auftrittszeitpunkts unvorhersagbar und zumeist nicht reproduzierbar
- Beispiele:
 - Signalisierung „normaler“ externer Ereignisse durch periphere Einheiten (Timer, Schalter, Grenzwertüber-/unterschreitung, ...)
 - Warnsignale der Hardware (Energienmangel, „Watchdog-timer“ abgelaufen, ...)
 - Beendigung einer Ein-/Ausgabeoperation (Wort von serieller Schnittstelle komplett empfangen oder komplett gesendet, Operation von Coprozessor (DMA, FPU) komplett, ...)
- Die Unterbrechungsbehandlung muß *nebeneffektfrei* verlaufen, d.h. das Hauptprogramm darf nach Abschluß der Behandlung keinen (ungewünschten) veränderten Ausführungs-Kontext vorfinden.
- Aber: typischerweise wird der Behandler zum Zwecke der Kommunikation über globale Variable mit dem Hauptprogramm kommunizieren.

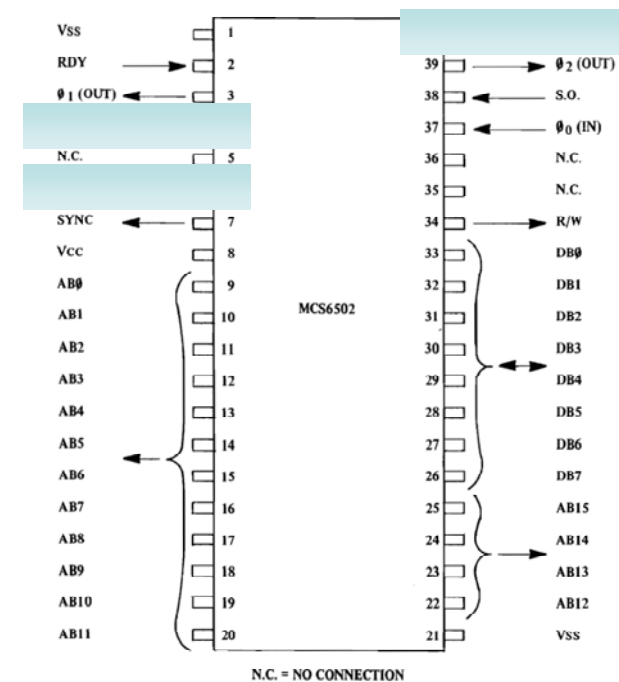
Beispiel: Modellfabrik (Praktikum) mit Prozessoren und INT-Quellen



Technische Realisierung von Interrupts

- Zur Realisierung besitzen Prozessoren einen oder mehrere spezielle Interrupt-Eingänge (typ. IRQ oder INT-Anschluß). Wird ein Interrupt aktiviert, so führt dies zur Ausführung einer Unterbrechungsbehandlungsroutine.
- Das Auslösen der Unterbrechungsroutine entspricht einem Unterprogrammaufruf. Der Programmablauf wird an einer anderen Stelle fortgesetzt und nach Beendigung der Routine (normalerweise) an der unterbrochenen Stelle fortgefahren.

Pin-Belegung des MOS 6502-Prozessors (z.B. Commodore 64).
Quelle: MCS 6500 HW-Manual, MOS Technology, Jan. 1976



Durchführung einer einfachen INT-Behandlung

1. IRQ-Anschluß wird durch peripheres Gerät aktiviert
2. Wenn Interrupts momentan zugelassen sind: Beendigung der Abarbeitung der gerade noch laufenden Instruktion
3. Sicherung der Register des Prozessors (Prozessorkontext) auf dem Stapelspeicher (Stack) – insbesondere Sicherung des Programmzählers; dafür spezielle Instruktionen verfügbar
4. Sprung an den Behandler (entspricht Laden des Programmzählers mit der Programmstartadresse des Behandlers) – dies kann auf verschiedene Arten erfolgen, siehe unten
5. Ausführung des Codes des Behandlers, an dessen Ende steht ein „Return from Interrupt“-Befehl; dabei Signalisierung an Peripherie, daß Behandlung abgeschlossen
6. Zurückladen des gesicherten Prozessorregistersatzes, Rücksprung (= Laden des Programmzählers mit der Adresse der Instruktion, die auf diejenige folgt, an der die Unterbrechung auftrat).

Beispiel INT beim 6502

- Bei Vorliegen eines IRQ wird der 16-Bit Programmzähler mit dem Inhalt der 8-Bit Adressen FFFE und FFFF geladen.
- FFFE/F enthält Adresse des (= *Vektor* auf) IRQ-Behandler
- FFFC/D enthält Adresse des Reset-Behandlers
- FFFA/B enthält Adresse des NMI-Behandlers. NMI: Non-Maskable-Interrupt, kann nicht abgeschaltet werden (also auch nicht durch fehlerhaftes Programm)

Speicher-Aufteilung für 6502 (in Hex-Adressen)

0000-00FF - RAM for Zero-Page & Indirect-Memory Addressing

0100-01FF - RAM for Stack Space & Absolute Addressing

0200-3FFF - RAM for programmer use

4000-7FFF - Memory mapped I/O

8000-FFF9 - ROM for programmer usage

FFFA - Vector address for NMI (low byte)

FFFB - Vector address for NMI (high byte)

FFFC - Vector address for RESET (low byte)

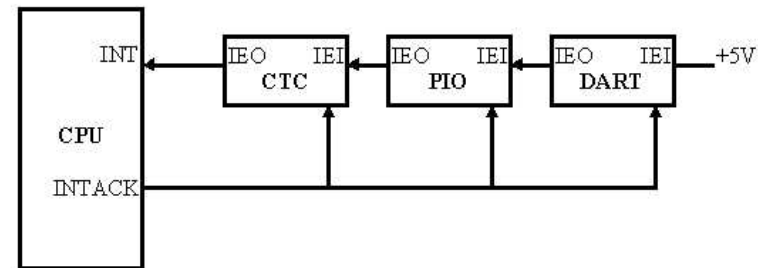
FFFD - Vector address for RESET (high byte)

FFFE - Vector address for IRQ & BRK (low byte)

FFFF - Vector address for IRQ & BRK (high byte)

Behandlung mehrerer Quellen von Interrupts

- Beim 6502 nur *ein* IRQ-Eingang und nur *ein* Vektor auf *einen* Behandler. Bei mehreren Quellen für Interrupts (z.B. serielle Schnittstelle, Parallele Schnittstelle, Timer) werden daher die Interrupt-Ausgänge dieser Einheiten „verodert“ und an den IRQ-Eingang gelegt. Alle Quellen sind damit gleichberechtigt.
- Damit ist keine automatische HW-Priorisierung (nach Wichtigkeit der eintreffenden Interrupts) möglich.
- Der Behandler muß nach dem Auftritt des Interrupts Quelle für Quelle abfragen, welche den Interrupt verursacht hat (implizite Priorisierung je nach Abfragereihenfolge)
- Einfache Möglichkeit der HW-Priorisierung: Daisy-Chain (Prioritätskette). Die Einheit, die am nächsten an der CPU liegt, hat die höchste Priorität (siehe Bild rechts) und sperrt Interrupts anderer Quellen solange aus, bis der eigene Interrupt abgearbeitet ist (INTACK durch Prozessor signalisiert).



Daisy-Chaining beim Z80-Prozessor
CTC: Counter-Timer-Circuit
PIO: Parallel In-/Out
DART: Dual Asynchronous Receiver Transmitter