Prozessorregister

Warum Register?

Speicher

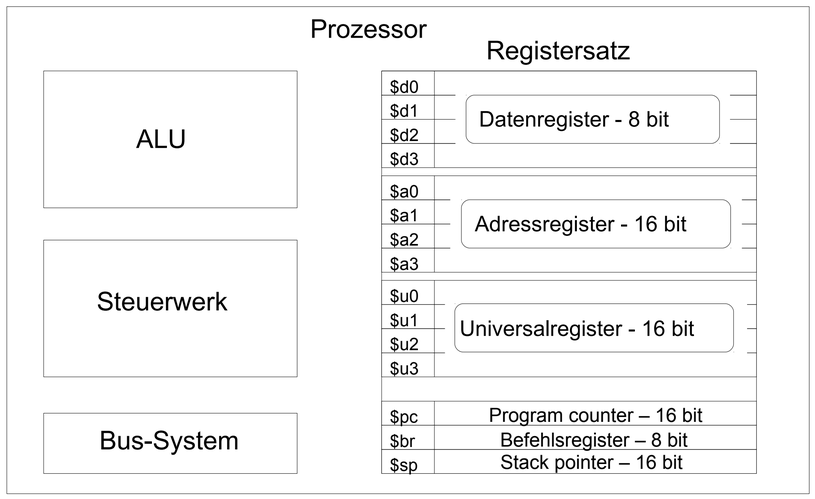
Arbeitsspeicher

Viel Speicherplatz
Längere Zugriffszeit
Adressierung durch Bus nötig

Prozessorregister

Wenig, beschränkter Speicher
Extrem kurze Zugriffszeit
Direktes Ansprechen möglich

Aufbau



Schema eines fiktiven 8-bit Prozessors

- Feste Bestandteile des Prozessors
- Geringe Anzahl
- Unterschiedliche Funktionen
- Anzahl, Größe, Typen stark von Architektur abhängig

Datenregister

- Direkt mit ALU verknüpft
- → Schnelle logische / arithmetische Operationen

Universalregister

- Verwendung dem Programmierer überlassen
→ Bessere Anpassung der Routine an Aufgabe

Befehlsregister

- Aktueller Befehl
→ Steuerwerk

Spezialregister

Sehr Prozessorspezifisch

- Effektive Adressierung von Speicherblöcken
- Angabe von Zuständen & Fehlern
- u. v. m.

Adressregister

- Adressierung des (Arbeits-)Speichers
→ Speichern / Laden über Adressbus
- Unterschiedliche Adressierungsformen

Program Counter (PC)

- Adresse aktueller / nächster Befehl
→ Korrekter Ablauf & Sprünge

Stackpointer

- Untere Adresse des Stacks
→ Stack = „Keller“ - von oben nach unten im Speicher
- Sichern: Dekrementieren um Datengröße → Push
- Wiederherstellen: Pop → Inkrementieren um Datengröße

Wortbreite

= „Größe“ eines Registers in Bits

- Evt. je nach Typ verschieden (s. Beispiel)
- Grundlegende Kategorisierung der Architektur (z.B. „32“ oder „64“ Bit)

Problem: Überlauf

Datenregister: Ergebnis der Operation größer als maximal mit Wortbreite darstellbar

→ Verfall des obersten Bits (oder mehrerer)

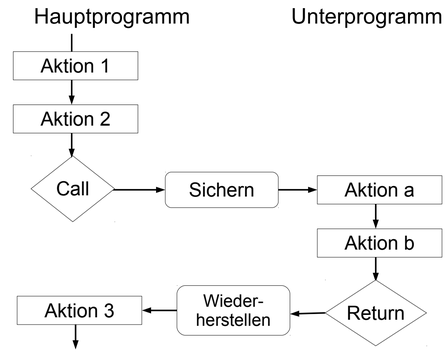
- Meistens „overflow-Bit“ der ALU zur Detektion

Quellen

- http://de.wikipedia.org/wiki/Register_%28Computer%29
- http://de.wikipedia.org/wiki/Prozessor
- http://de.wikipedia.org/wiki/Stapelspeicher
- http://de.wikipedia.org/wiki/Interrupt

Unterprogramme

- Häufiges Auftreten des selben Teilproblems
→ Auslagerung der Routine
- Effizientere Lösung durch weniger Code (kein „Spaghetti-Code“)
- Haben i.d.R. die selben Ressourcen (Register) zur Verfügung wie das Hauptprogramm



Externe Ereignisse

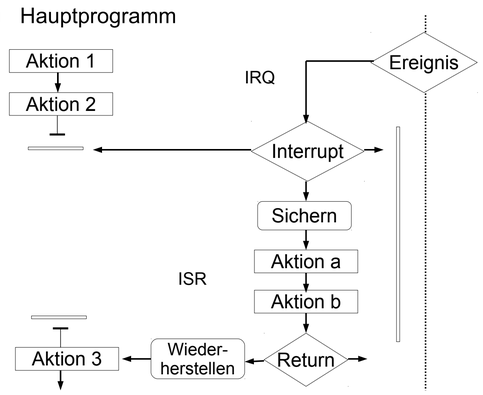
- z.B. Tastendruck, Netzwerkereignis, Timer, …
- Entstehung unabhängig vom Programm
- Erkennung & Behandlung durch laufenden Routine

Behandlung

Polling	Interrupt
synchron → planbar Andauernder Prozess Zeit- & Ressourcen-intensiv	asynchron → unvorhersehbar Einmaliger Prozess Zeit- & Ressourcen-schonend

Interrupts

- Unterbrechung des aktuellen Programmablaufes
- Meist direkt durch Interrupt-Pin am Prozessor
- Behandeln zeitkritische Ereignisse
→ Schnell & kurz



- Interrupt request (IRQ)
- Andere Interrupts nach Priorität blockieren
- Sicherung des aktuellen Zustandes
- Abarbeiten der Service-Routine (ISR)
- Wiederherstellung des alten Zustandes
- Blockierte Interrupts zulassen
- Zur vorherigen Routine zurückspringen

Zustandssicherung - Stack

Wieso sichern?

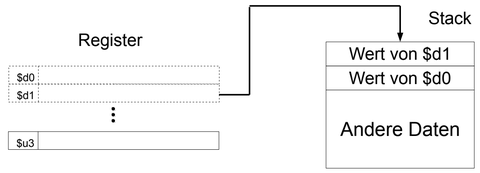
Bei Interrupts & Unterprogrammen

- Für Unterroutine sollen alle Register zur Verfügung stehen
→ Auslagern der gespeicherten Daten
- Alle Register besitzen danach wieder alten Wert
→ Für Hauptroutine keine Veränderung

→ Tiefe Verschachtelungen von Routinen möglich (z.B. Rekursion)

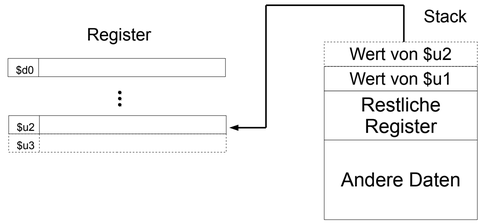
Push

Legt ein Element „oben“ auf den Stapel



Pop

Nimmt ein Element „von oben“ vom Stapel herunter



Sichern: „Push“ aller Registerinhalte auf den Stack

Wiederherstellen:„Pop“ der Daten vom Stack in die Register

Jens Kreber