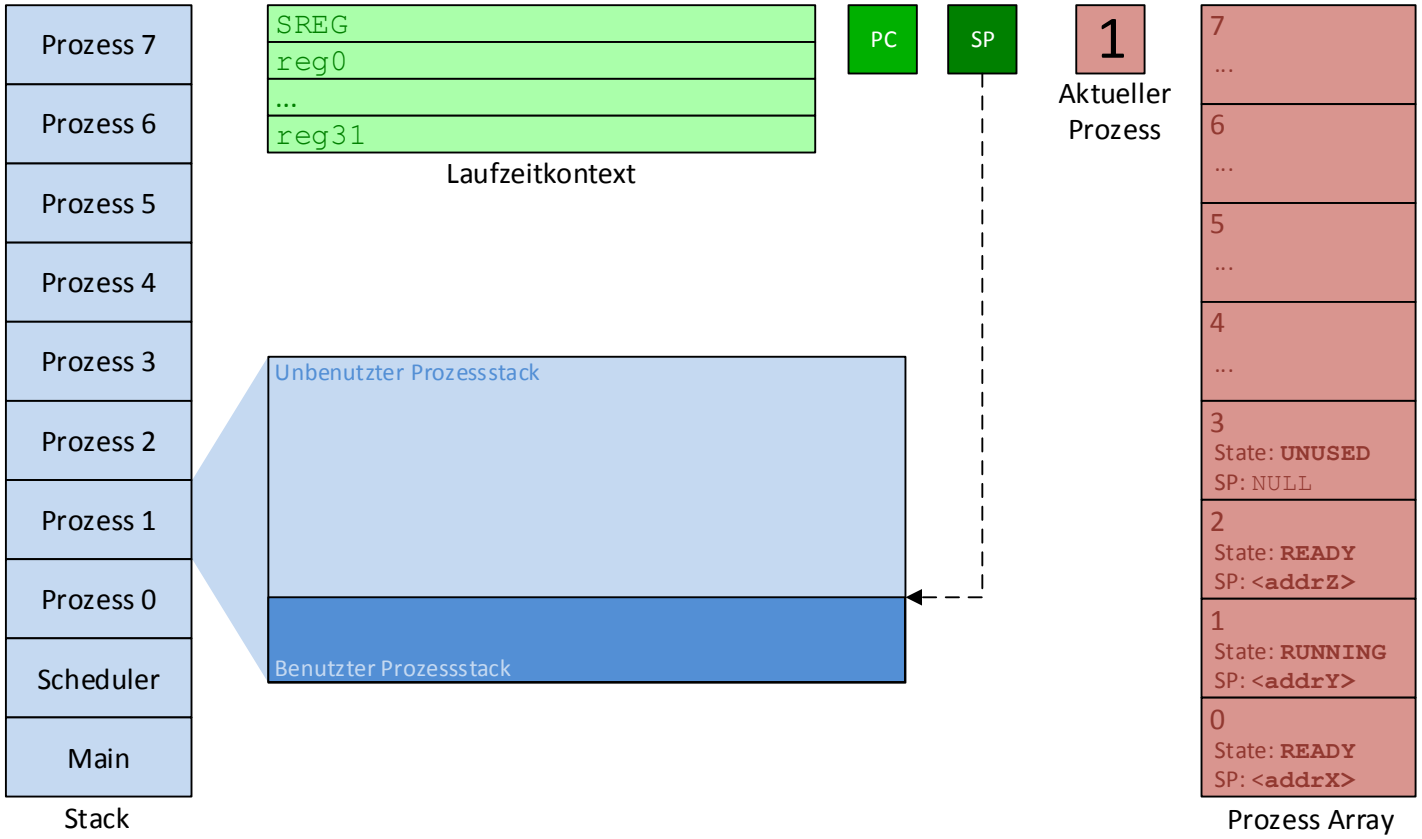
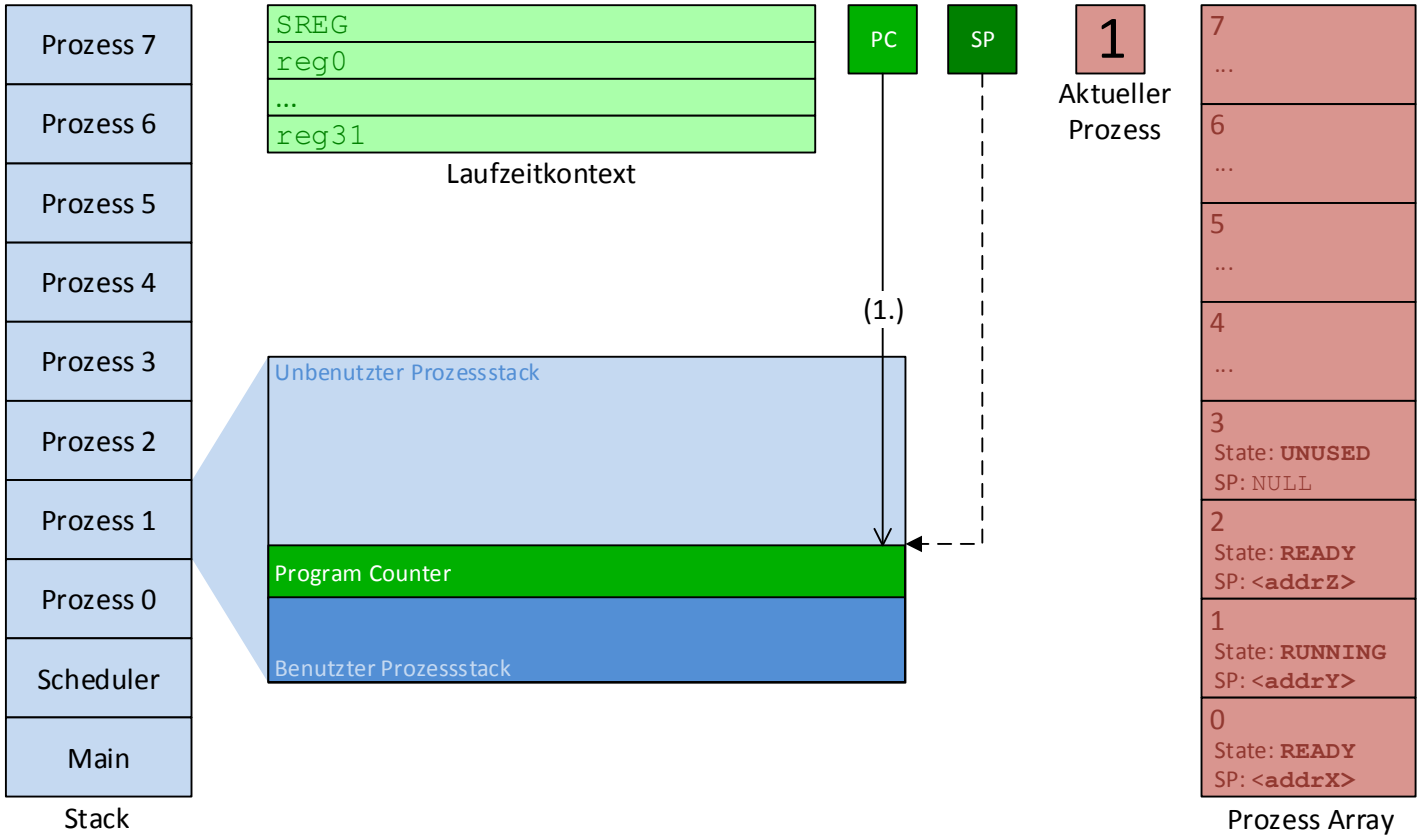
Anfangszustand: Prozesstack von Prozess 1



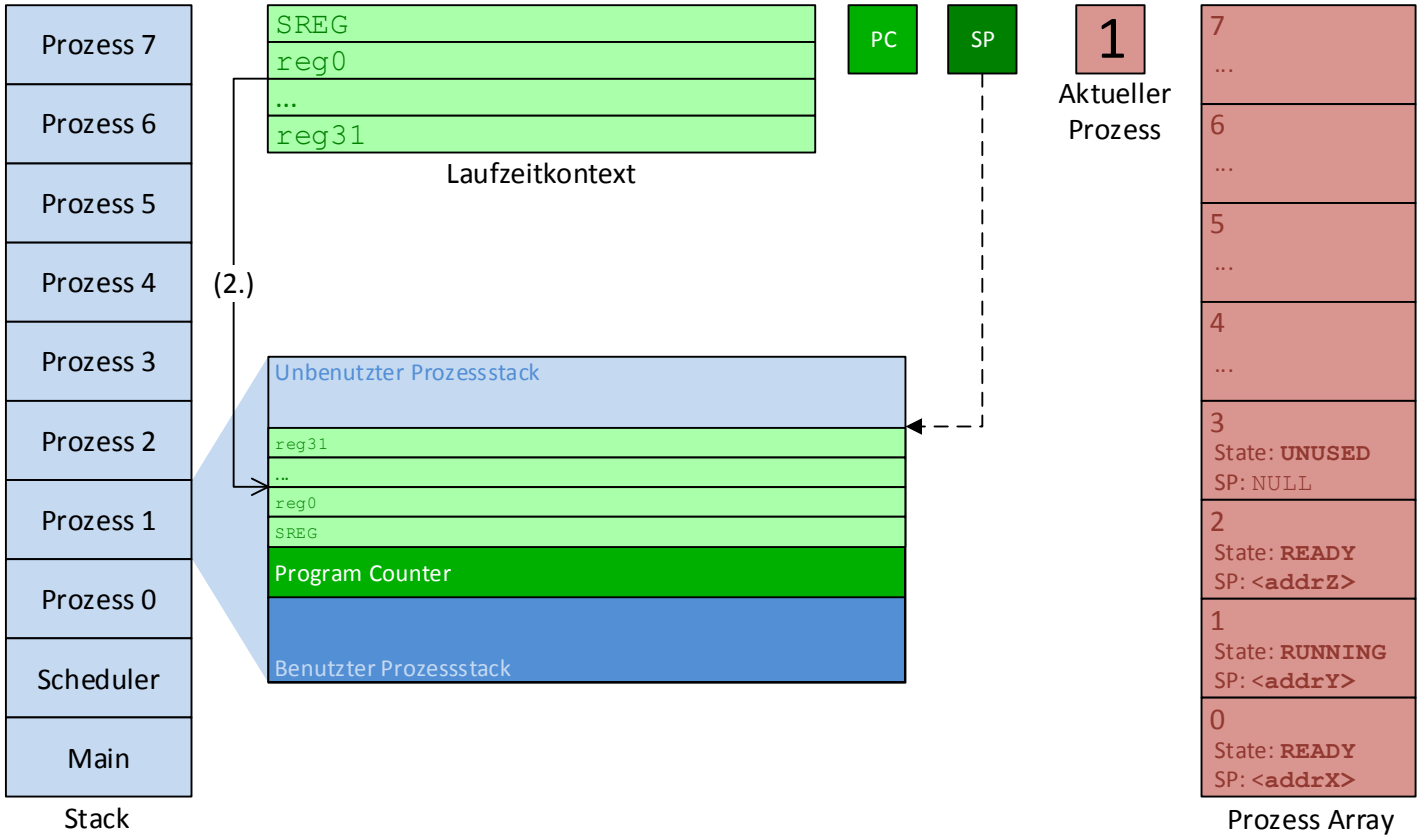
Die obige Abbildung illustriert den Zustand von SPOS vor dem Prozesswechsel, wobei der Stack rechts, Register des Mikrocontrollers oben und Datenstrukturen zur Prozessverwaltung links angeordnet sind.

Schritt 1: Speichern des Programmzählers



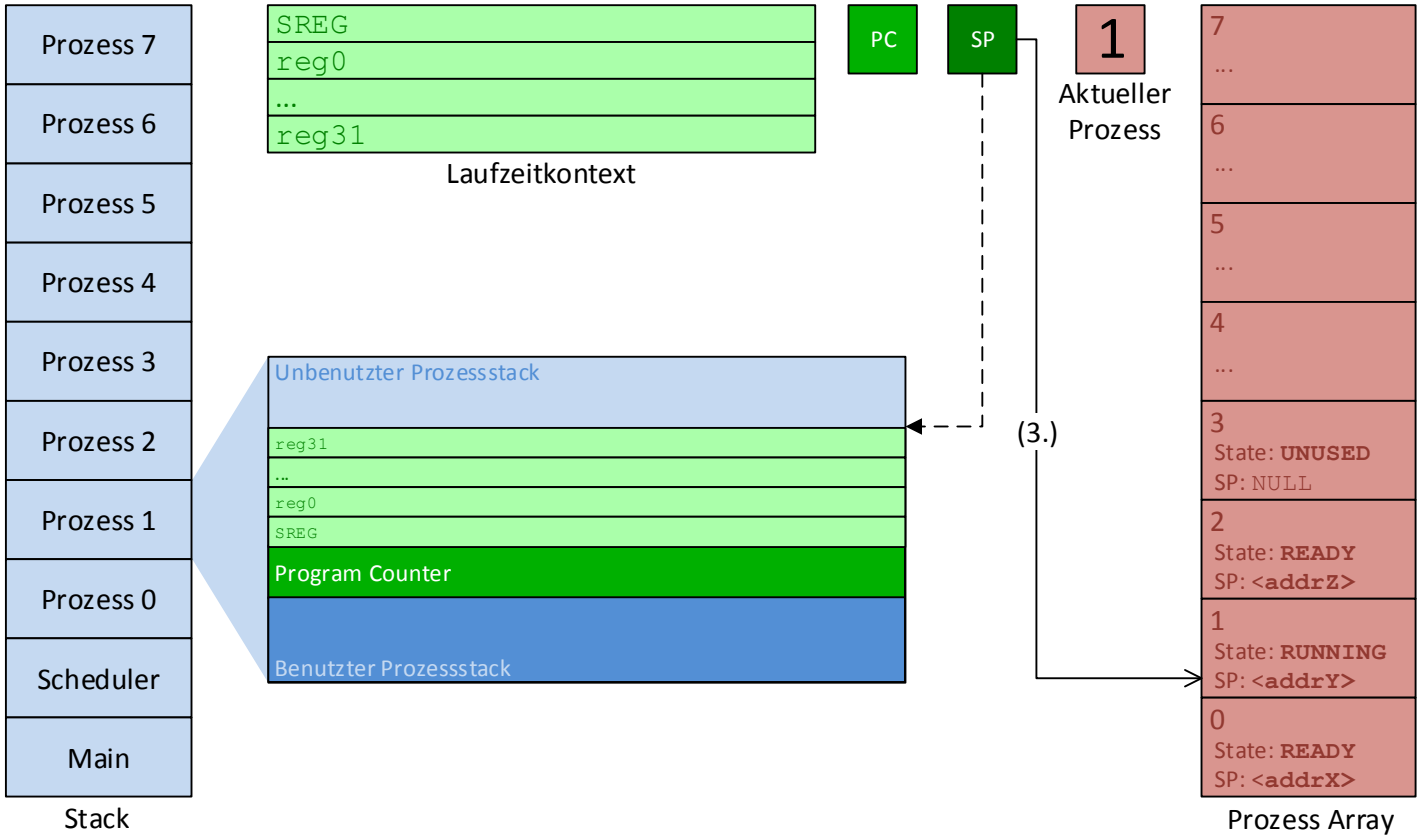
Beim Sprung in die ISR wird implizit der Programmzählers als Rücksprungadresse für die spätere Fortsetzung des aktuellen Prozesses auf dessen Prozessstack gespeichert. Der Stackpointer zeigt anschließend auf die erste freie Stelle im Prozessstack.

Schritt 2: Sichern des Laufzeitkontextes



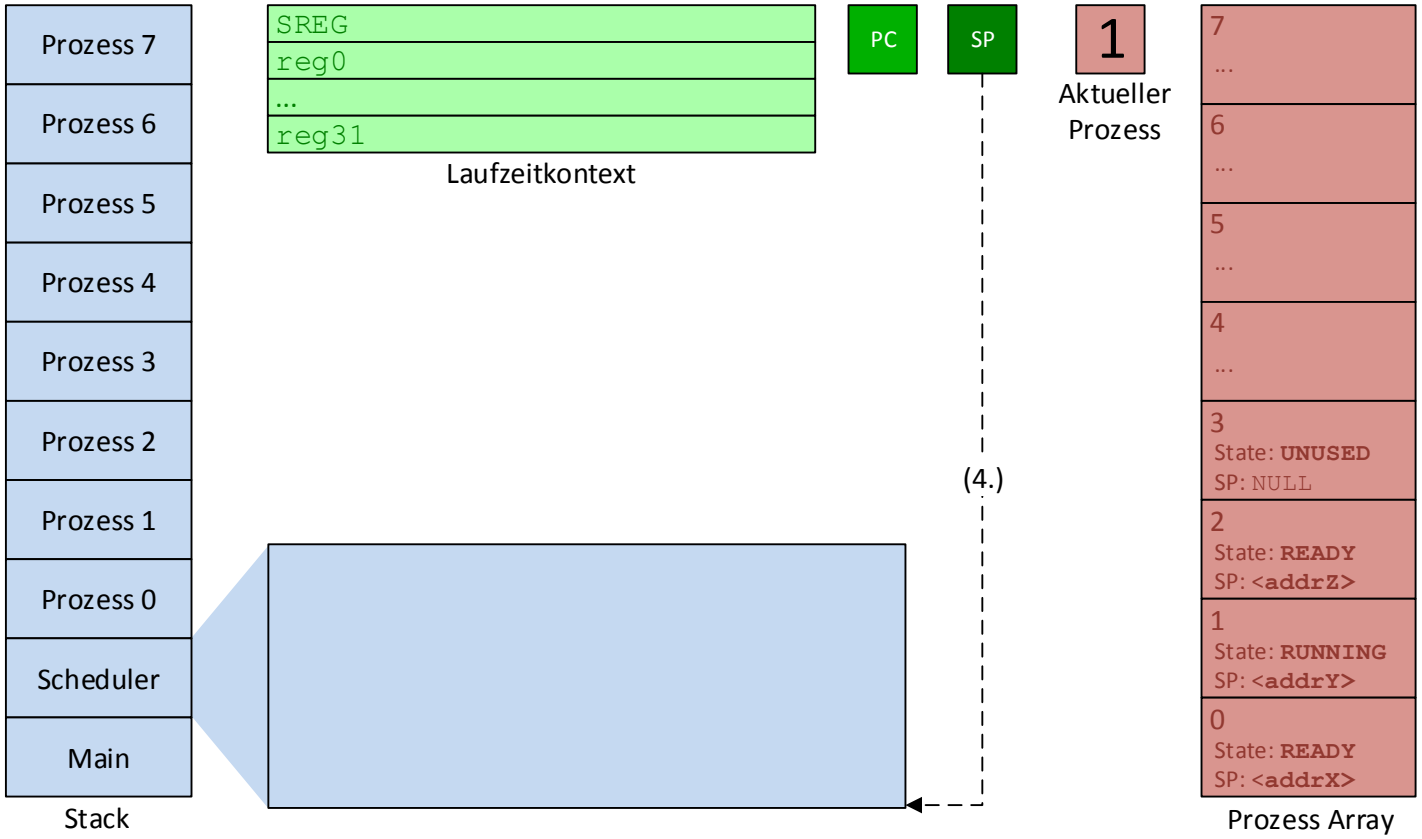
Durch Aufruf des Assemblermakros `saveContext()` wird der Laufzeitkontext zum Zeitpunkt der Unterbrechung durch den Scheduler auf dessen Prozessstack gesichert. Anschließend zeigt der Stackpointer auf die erste freie Stelle im Prozesstack.

Schritt 3: Sichern des Stackpointers



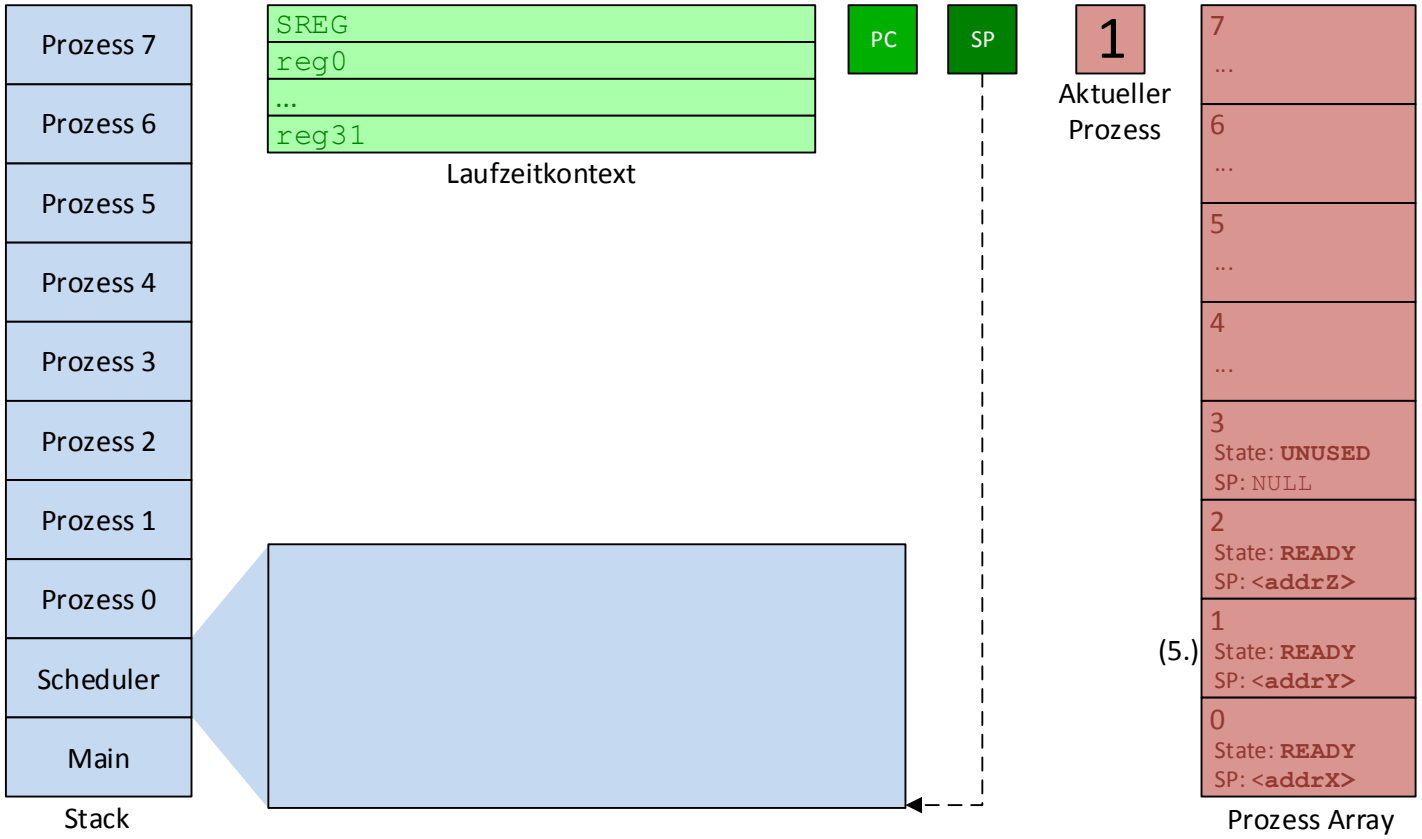
Zur späteren Wiederherstellung des Laufzeitkontextes für Prozess 1 wird die aktuelle Adresse des Stackpointers im Prozess-Array gesichert.

Schritt 4: Setzen des Stackpointers auf den ISR-Stack



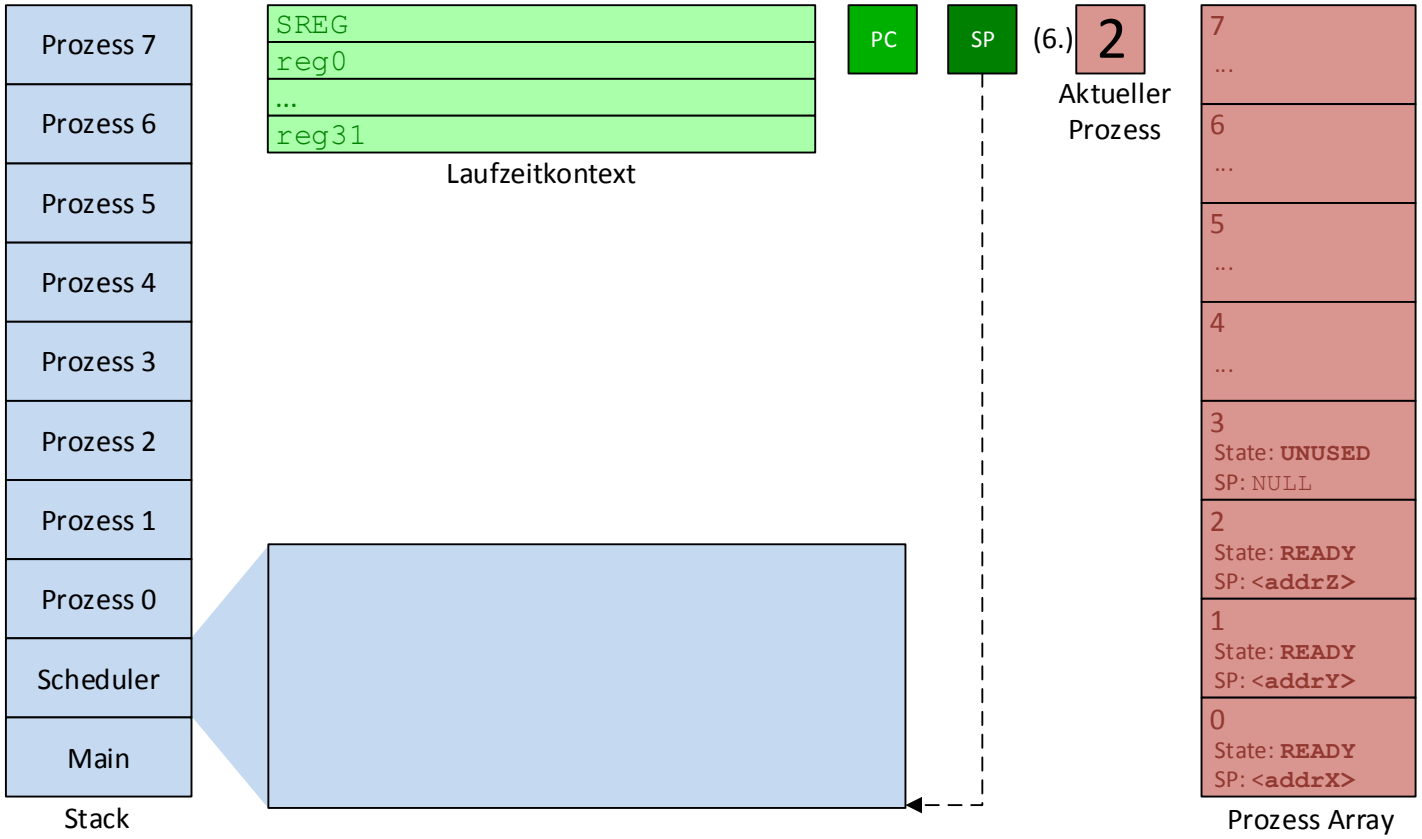
Zur eigentlichen Ausführung der Scheduler-ISR wird der Stackpointer auf den Anfang des ISR-Stacks gesetzt.

Schritt 5: Aktueller Prozesses auf `OS_PS_READY`



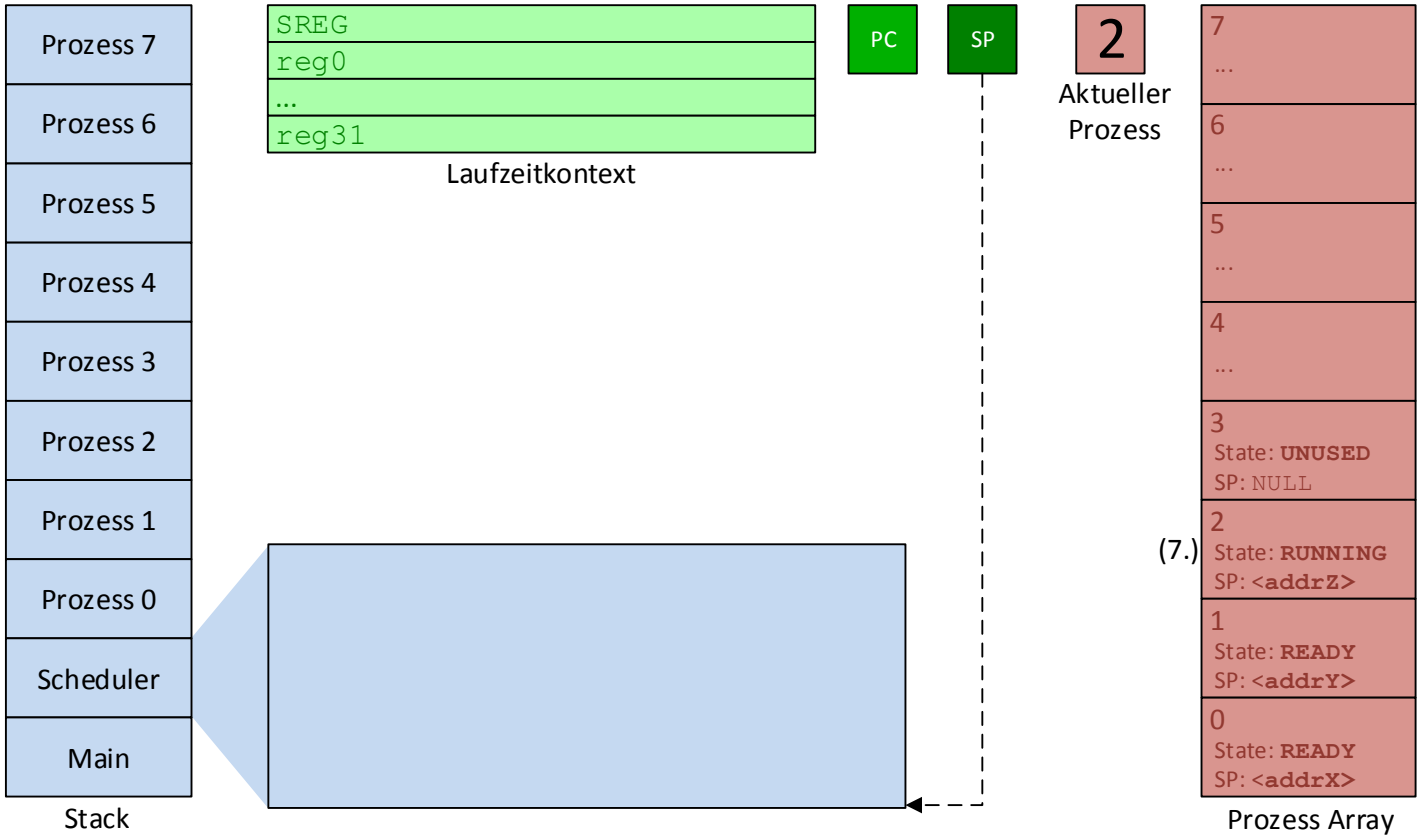
Der Prozesszustand des aktuellen Prozesses wird im Prozess-Array auf `OS_PS_READY` gesetzt.

Schritt 6: Auswahl nächsten Prozesses



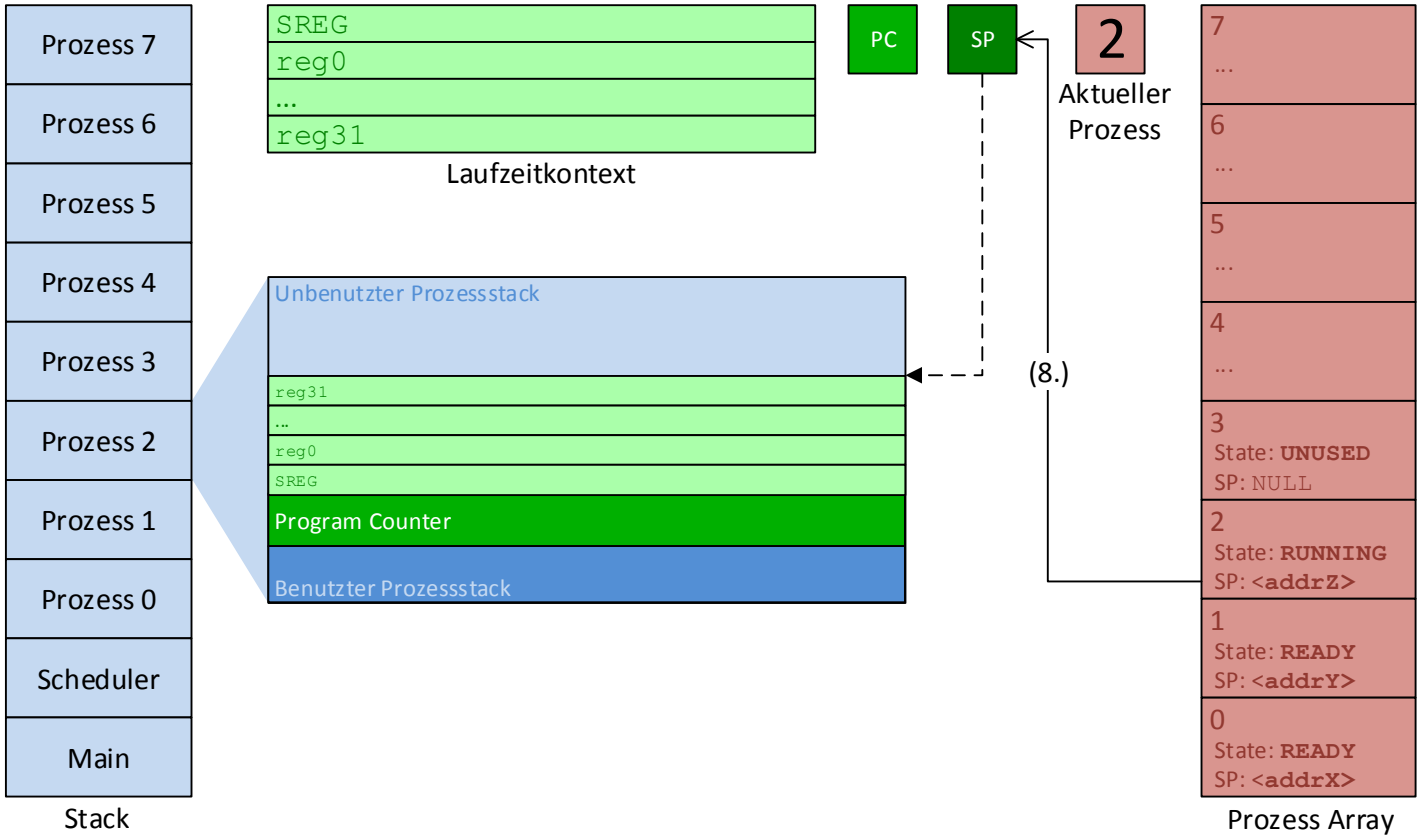
Mit Hilfe der aktuell verwendeten Schedulingstrategie wird der nächste fortzusetzende Prozess ausgewählt, der in unserem Beispiel Prozess 2 ist.

Schritt 7: Nächster Prozess auf `OS_PS_RUNNING`



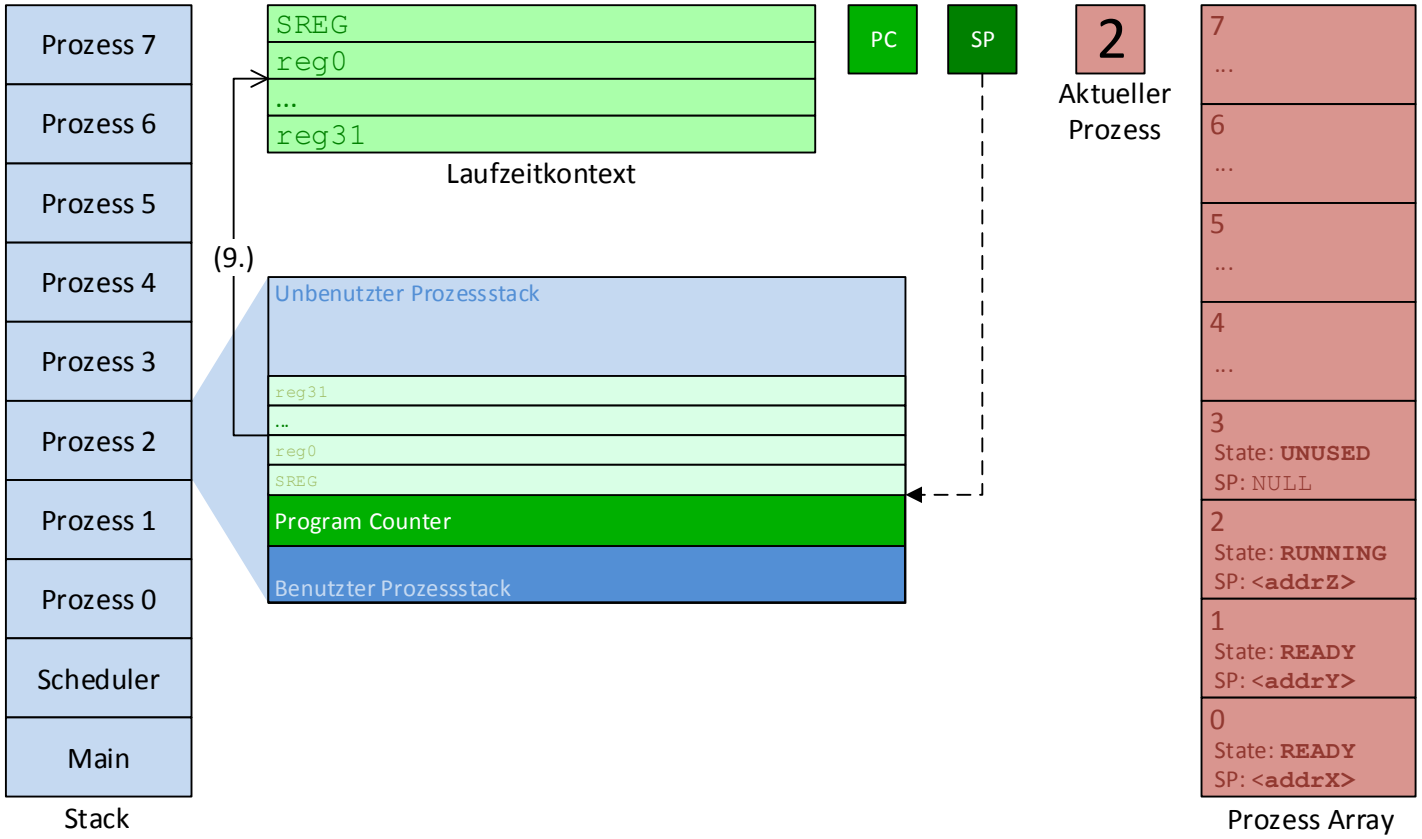
Nachdem Prozess 2 als nächster fortzusetzender Prozess ausgewählt wurde, wird dessen Prozesszustand im Prozess-Array auf `OS_PS_RUNNING` gesetzt.

Schritt 8: Wiederherstellen des Stackpointers



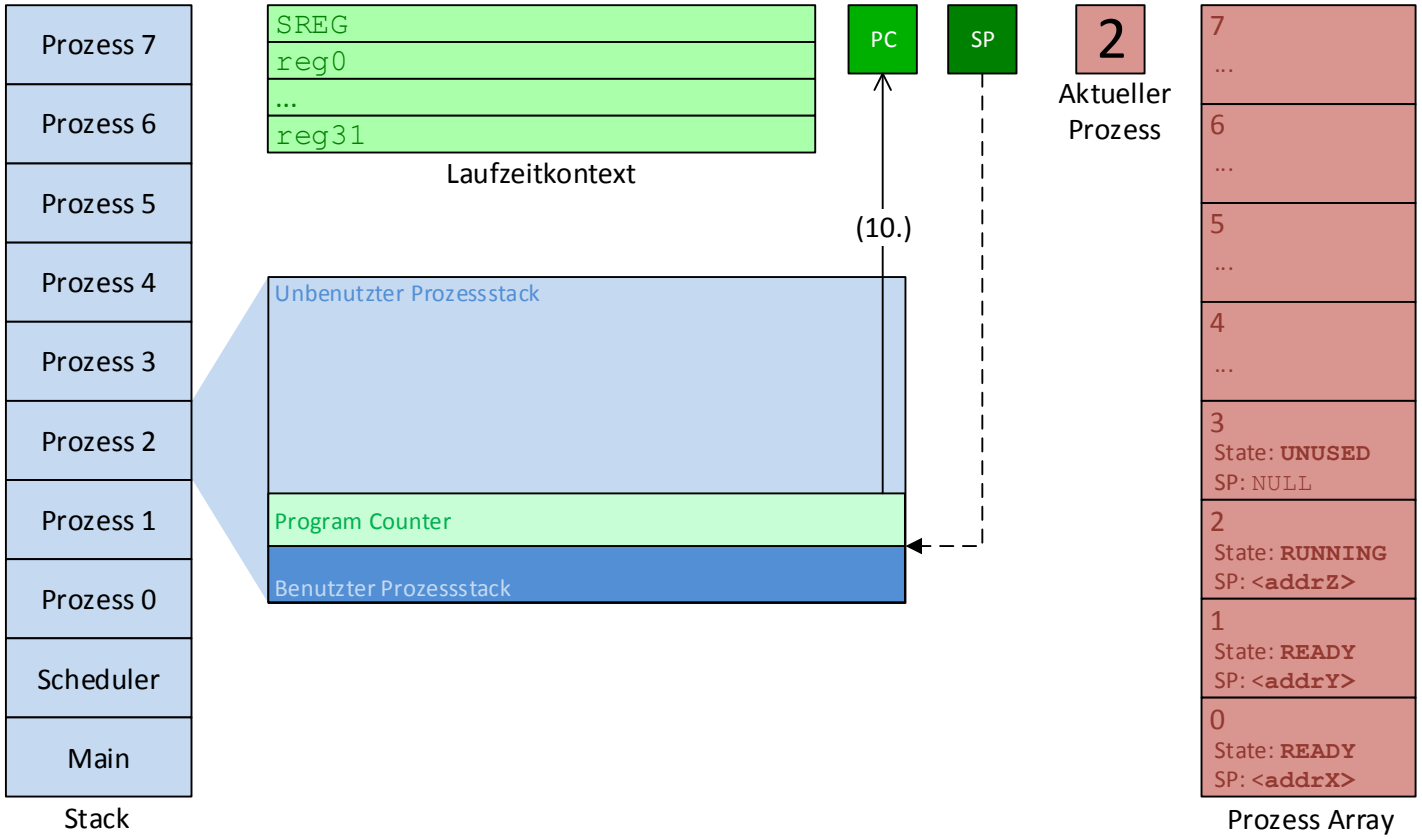
Die zu einem früheren Zeitpunkt im Prozess-Array gesicherte Adresse des Stackpointers, die auf die erste freie Stelle oberhalb des Laufzeitkontextes, Program Counters und des restlichen benutzten Prozesstacks von Prozess 2 zeigt, wird in diesem Schritt wiederhergestellt. Dieser Schritt ist als Äquivalent zu Schritt 3 *Sichern des Stackpointers* zu sehen.

Schritt 9: Wiederherstellen des Laufzeitkontextes



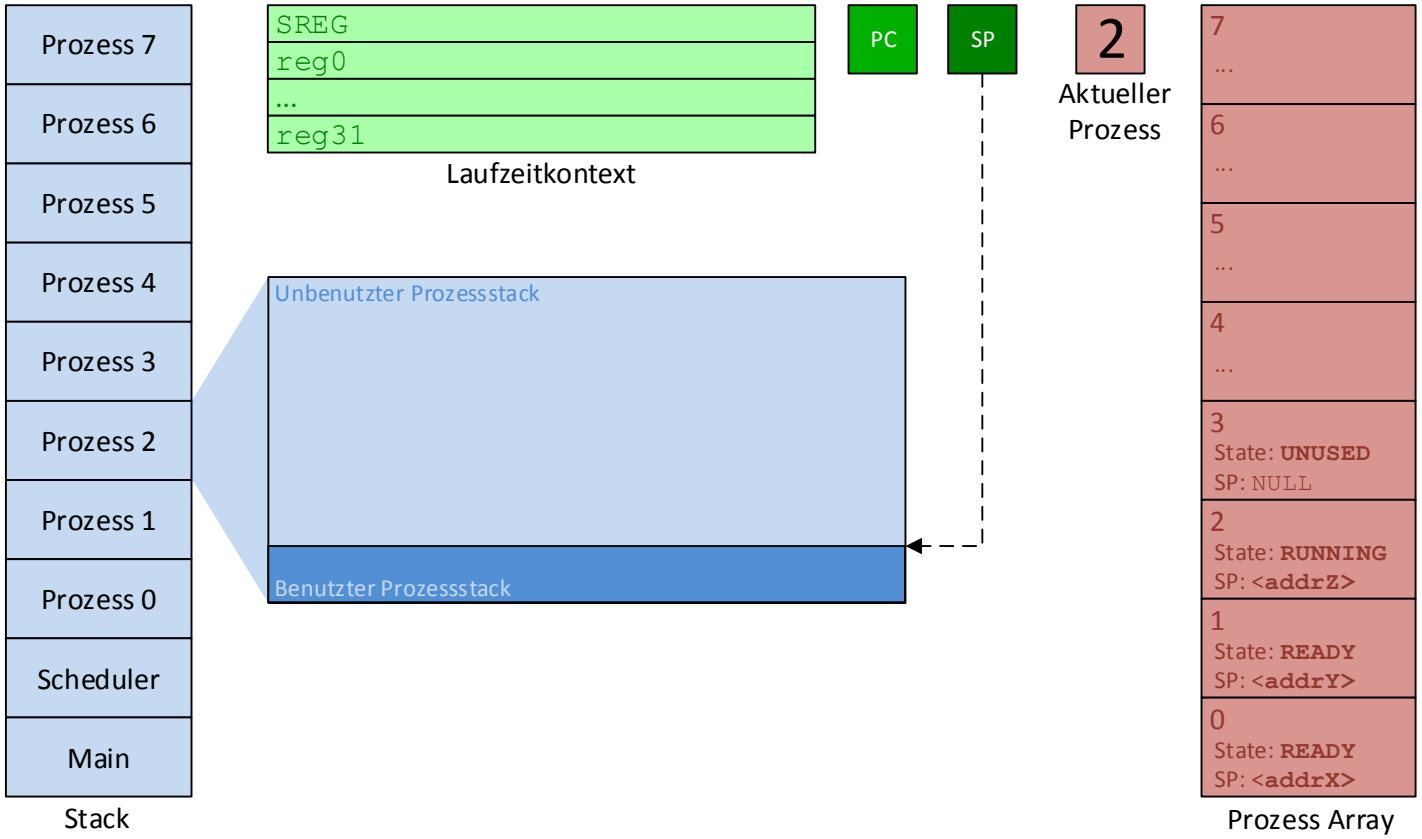
Durch das Assemblermakro `restoreContext()` wird der zu einem früheren Zeitpunkt gesicherte Laufzeitkontextes von Prozess 2 wiederhergestellt. Dieser Schritt ist als Äquivalent zu Schritt 2 *Sichern des Laufzeitkontextes* zu sehen.

Schritt 10: Rücksprung



Durch den im Assemblermakro `restoreContext()` enthaltene Befehl `reti` erfolgt ferner ein automatischer Rücksprung an die Stelle im Programablauf, an der der Prozess zu einem früheren Zeitpunkt vom Scheduler unterbrochen wurde. Dieser Schritt ist als Äquivalent zu Schritt 1 *Speichern des Programmzählers* zu sehen.

Endzustand: Prozessstack von Prozess 2



Nun steht dem fortgesetzten Prozess der Laufzeitkontext zur Verfügung, den dieser vor der Unterbrechung durch den Scheduler hatte. Die Ausführung wird an der Stelle fortgesetzt, an der sie zuvor durch den Scheduler unterbrochen wurde, da die zuvor auf dem Stack abgelegten Register und der Programmzähler wieder hergestellt worden sind.