(19) Bundesrepublik Deutschland
Deutsches Patent- und Markenamt

(10) **DE 10 2008 020 782 A1** 2009.11.12

(12) # Offenlegungsschrift

(21) Aktenzeichen: **10 2008 020 782.9**
(22) Anmeldetag: **25.04.2008**
(43) Offenlegungstag: **12.11.2009**

(51) Int Cl.[8]: ***G06F 9/46*** (2006.01)
G06F 9/38 (2006.01)
G06F 9/48 (2006.01)

(71) Anmelder:
Fujitsu Siemens Computers GmbH, 80807 München, DE

(74) Vertreter:
Epping Hermann Fischer, Patentanwaltsgesellschaft mbH, 80339 München

(72) Erfinder:
Groß, Jürgen, 83026 Rosenheim, DE

(56) Für die Beurteilung der Patentfähigkeit in Betracht gezogene Druckschriften:

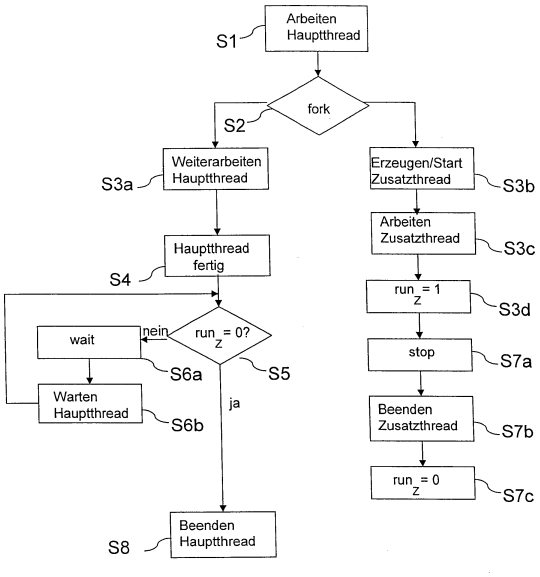
Wilson,R.: Embedded core adds multithread. EE Times. 13.10.2003. <http://www.eetimes.com/news/design/silicon/showArticle.jhtml?articleID=174087588&kc=6325> (recherchiert am 18.12.08)

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

Prüfungsantrag gemäß § 44 PatG ist gestellt.

(54) Bezeichnung: **Verfahren zur parallelen Verarbeitung von Programmen sowie zur Durchführung des Verfahrens geeigneter Prozessor**

(57) Zusammenfassung: Die Erfindung betrifft einen Prozessor, der zur parallelen Ausführung von wenigstens zwei Threads mittels Hyper-Threading auf wenigstens zwei logischen Prozessoren geeignet ist. Beim Prozessor ist ein Prozessorbefehl (fork) vorgesehen, der dazu geeignet ist, einen zweiten Thread aus einem ersten Thread heraus zu erzeugen und/oder zu starten, sowie ein Prozessorbefehl (stop), der dazu geeignet ist, einen aktuellen zweiten Thread zu beenden.
Die Erfindung betrifft ebenso ein Verfahren zu parallelen Verarbeitung von Programmen durch den erfindungsgemäßen Prozessor.



Beschreibung

[0001] Die Erfindung betrifft einen Prozessor der zur parallelen Ausführung von wenigstens zwei Threads mittels Hyper-Threading auf wenigstens zwei logischen Prozessoren geeignet ist sowie ein Verfahren zur parallelen Verarbeitung von Programmen durch den erfindungsgemäßen Prozessor.

[0002] Ein Prozessor ist eine Recheneinheit eines elektronischen Geräts (beispielsweise eine CPU (Central Processing Unit) in einem Computersystem), die über eine geeignete Software – im allgemeinen ein Betriebssystem – Komponenten des Geräts steuert. Eine grundlegende Eigenschaft des Prozessors ist seine Programmierbarkeit. Die Betriebsweise des Prozessors wird von Programmen in Form eines Maschinencodes bestimmt. Hauptbestandteile des Prozessors sind Register – als Speicher für Zwischenergebnisse –, ein Rechenwerk (auch bezeichnet als arithmetisch-logische Einheit (ALU)) sowie ein Steuerwerk, das durch Setzen geeigneter Signale die übrigen Bestandteile des Prozessors steuert.

[0003] Um die Leistungsfähigkeit eines Prozessors zu steigern, wird bei modernen Prozessoren oftmals das so genannte Hyper-Threading, also das parallele Abarbeiten von so genannten Threads, angewandt. Ein Thread ist dabei ein Teil eines abzuarbeitenden Programms (d. h. ein Programmabschnitt), den der Prozessor auf einmal bearbeiten kann. Die Hyper-Threading Technologie ermöglicht einem einzelnen physikalischen Prozessor zwei separate Programmabschnitte gleichzeitig zu bearbeiten, wodurch die Auslastung des Prozessors erhöht wird und eine höhere Leistungsfähigkeit erzielt wird.

[0004] Beim Hyper-Threading ist der physikalische Prozessor in zwei oder mehr logische Prozessoren aufgespalten, wobei sich die beiden logischen Prozessoren einen Teil der physikalischen Ausführungs-Ressourcen (Cache, System-Bus-Interface) teilen. Die Umsetz-Logik, wie beispielsweise Registersätze oder ein Unterbrechungs-Controller (Advanced Programmable Interrupt Controller (APIC)), der Unterbrechungsaufforderungen von Geräten im System entgegennimmt, ist für jeden logischen Prozessor separat vorhanden. Die logischen Prozessoren arbeiten parallel und weitgehend unabhängig voneinander, was den Datendurchsatz steigert und die Auslastung des physikalischen Prozessors erhöht, wodurch dessen Ressourcen besser ausgenutzt werden.

[0005] Ein einzelnes Programm läuft durch die Nutzung des Hyper-Threadings nur dann schneller ab, wenn es in sinnvoller und zweckmäßiger Weise Aufgaben auf zwei oder mehrere Threads – abhängig von der Anzahl der logischen Prozessoren – aufteilen kann. Durch eine ungünstige Einteilung der Threads bzw. ein nicht sinnvolles Belegen der Threads mit Teilaufgaben/Programmabschnitten kann es vorkommen, dass die Gesamtlaufzeit des Programms durch das Hyper-Threading länger wird, als wenn es seriell in einem einzelnen Thread abgearbeitet wird. Nach dem Stand der Technik ist die Parallelisierung eines Programms in mehrere Threads daher auch nur dann sinnvoll, wenn jeder Thread hinreichend lang arbeiten kann, die Programmabschnitte also ausreichend groß sind.

[0006] Aufgabe der Erfindung ist es daher einen zur parallelen Ausführung von wenigstens zwei Threads mittels Hyper-Threading auf wenigstens zwei logischen Prozessoren geeigneten Prozessor zu beschreiben, durch den die Threads entsprechend den Anforderungen eines Programms zur Abarbeitung von Programmabschnitten eingeteilt werden und durch den auch kurze Programmabschnitte effizient und durchsatzsteigernd parallelisiert werden können. Zudem ist es Aufgabe der Erfindung ein Verfahren zur parallelen Verarbeitung von Programmen zu beschreiben, mit dem ein Datendurchsatz auch bei der Verarbeitung nur kurzer Programmabschnitte erheblich gesteigert wird.

[0007] Die Aufgabe wird erfindungsgemäß durch einen Prozessor der eingangs genannten Art gelöst, der dadurch gekennzeichnet ist, dass ein Prozessorbefehl vorgesehen ist, der dazu geeignet ist einen zweiten Thread zu erzeugen und/oder zu starten und dass ein weiterer Prozessorbefehl vorgesehen ist, der dazu geeignet ist einen aktuellen zweiten Thread zu beenden.

[0008] Die Erfindung liegt dabei in den zusätzlichen Prozessorbefehlen, die ermöglichen einen zweiten Thread bedarfsgerecht auf Hardware Ebene aus einem ersten Thread heraus zu erzeugen bzw. auch wieder zu beenden und dafür ohnehin vorhandene Ressourcen des Prozessors (Speicher, Registerwerke) auszunutzen. Dadurch können neue Threads abhängig von Anforderungen eines Benutzerprogramms erzeugt werden, was die parallele Ausführung auch kleinster Programmabschnitte nicht nur ermöglicht, sondern auch effizienter macht und die Leistungsfähigkeit des Prozessors erheblich steigert.

[0009] In einer bevorzugten Ausgestaltung der Erfindung ist ein weiterer Prozessorbefehl vorgesehen, der dazu geeignet ist, dass der erste Thread wartet, während der zweite Thread noch arbeitet, bzw. der zweite Thread wartet, während der erste Thread noch arbeitet. Durch diesen Prozessorbefehl wird beispielsweise erreicht, dass ein erster/zweiter Thread erst dann beendet wird, wenn der jeweils andere mit der Abarbeitung eines Programmabschnittes fertig ist, was die Synchronisierung der Threads erleichtert.

[0010] In einer ebenso bevorzugten Ausgestaltung der Erfindung ist ein Prozessorbefehl vorgesehen, der dazu geeignet ist, den zweiten Thread anzuhalten und ein oder mehrere Register des zweiten Threads zu speichern. Zudem ist ein weiterer Prozessorbefehl vorgesehen, der dazu geeignet ist, den mittels des ersten Prozessorbefehls angehaltenen zweiten Thread mit einem vorgegebenen Registerinhalt wieder zu starten. Dies ermöglicht ein Anhalten und erneutes Starten des zweiten Threads, beispielsweise bei der Behandlung einer Unterbrechung.

[0011] In einer weiteren bevorzugten Ausgestaltung der Erfindung ist ein erster Registersatz für den auf einem ersten logischen Prozessor ablaufenden ersten Thread sowie ein zweiter Registersatz für den auf einem zweiten logischen Prozessor ablaufenden zweiten Thread vorgesehen.

[0012] In einer bevorzugten Ausführung ist eine Zuordnung von zwei Zustandsbits für jedes Register des zweiten Registersatzes vorgesehen. Die Zustandsbits sind dazu eingerichtet anzuzeigen, ob ein Register des zweiten Registersatzes einen gültigen Inhalt hat und ob ein Register des ersten Registersatzes den selben Wert enthält, wie das entsprechende Register des zweiten Registersatzes. Dadurch kann eine Abstimmung bzw. Angleichung von Registerinhalten der Threads wesentlich erleichtert werden.

[0013] In einer ebenso bevorzugten Ausführung der Erfindung ist bei einer Initialisierung des Prozessors eine Initialisierung des ersten Registersatzes sowie des zweiten Registersatzes mit denselben Werten vorgesehen.

[0014] Die Aufgabe wird ebenso erfindungsgemäß durch ein Verfahren der eingangs genannten Art gelöst. Bei dem Verfahren wird ein auf einem ersten logischen Prozessor ablaufender erster Thread durch den erfindungsgemäßen Prozessor zur Abarbeitung eines Programms definiert und gestartet. Mittels eines Prozessorbefehls wird daraufhin ein zweiter Thread aus dem ersten Thread heraus erzeugt und auf einem zweiten logischen Prozessor gestartet. Der zweite Thread wird durch einen weiteren Prozessorbefehl wieder gestoppt. Anschließend wird der erste Thread durch den Prozessor beendet und/oder es wird eine Abarbeitung eines neuen Programms in dem ersten Thread durch den Prozessor gestartet.

[0015] Die Erfindung liegt dabei wiederum in den bereits oben beschriebenen Prozessorbefehlen des erfindungsgemäßen Prozessors zur parallelen Verarbeitung auch kleiner Programmabschnitte. Der zweite Thread wird dabei dynamisch aus dem ersten Thread heraus erzeugt. Durch das Verfahren wird der Prozessor besser ausgelastet, die Threads werden abhängig von den Anforderungen des Programms zur Abarbeitung von Programmabschnitten angewiesen und dadurch ein höherer Datendurchsatz – insbesondere bei der parallelen Verarbeitung nur kurzer Programmabschnitte – erreicht.

[0016] In einer bevorzugten Ausführung der Erfindung läuft der zweite Thread mit denselben Registerwerten an, die der erste Thread beim Abgeben des Prozessorbefehls zum Erzeugen/Starten des zweiten Threads aufweist. Dies macht ein zeitaufwändiges Laden von neuen Registerwerten für den zweiten Thread bei dessen Erzeugung/Start überflüssig.

[0017] In einer ebenso bevorzugten Ausgestaltung werden die Registerwerte eines zweiten dem zweiten Thread zugeordneten Registersatzes, während der zweite Thread nicht läuft, kontinuierlich auf dem selben Stand gehalten, wie die Registerwerte eines ersten dem ersten Thread zugeordneten Registersatzes. Dadurch ist ein sofortiges Anlaufen des zweiten Threads nach dessen Erzeugung garantiert.

[0018] Eine weitere bevorzugte Ausführung der Erfindung sieht vor, dass der erste Thread und der zweite Thread sich lediglich durch ein Flag oder durch einen Programmzähler unterscheiden. Aufwändige Umgestaltungen oder Erweiterungen des Programmcodes zur Unterscheidung der beiden Threads entfallen damit.

[0019] Weitere Einzelheiten und Ausgestaltungen der Erfindung sind in den Unteransprüchen angegeben.

[0020] Die Erfindung wird nachfolgend an Ausführungsbeispielen anhand der Zeichnungen näher erläutert.

[0021] In den Zeichnungen zeigen:

[0022] Fig. 1 ein Ablaufdiagramm für ein Verfahren zur parallelen Verarbeitung von Programmen mittels des erfindungsgemäßen Prozessors,

[0023] Fig. 2 ein Ablaufdiagramm für ein Verfahren zur parallelen Verarbeitung von Programmen mittels des erfindungsgemäßen Prozessors in einem weiteren Ausführungsbeispiel,

[0024] Fig. 3 ein Ablaufdiagramm für einen Teilbereich des in Fig. 1 dargestellten Verfahrens,

[0025] Fig. 4 ein Ablaufdiagramm für einen weiteren Teilbereich des in Fig. 1 dargestellten Verfahrens.

[0026] Fig. 1 zeigt ein Ablaufdiagramm für ein Verfahren zur parallelen Verarbeitung von Programmen in zwei Threads durch Hyper-Threading auf zwei logischen Prozessoren mittels der erfindungsgemäßen Prozessors.

[0027] Schritt S1 beschreibt dabei das Abarbeiten eines Programms/Programmabschnitts in einem auf einem ersten logischen Prozessor ablaufenden ersten Thread, der im folgenden als Hauptthread bezeichnet wird. Dabei ist, abhängig von der Zahl der logischen Prozessoren im System, selbstverständlich nicht nur ein Hauptthread – wie hier dargestellt – sondern auch eine größere Anzahl von Hauptthreads vorstellbar.

[0028] In Schritt S2 ergeht ein Prozessorbefehl (fork), durch den ein zweiter Thread, der im folgenden als Zusatzthread bezeichnet wird, dynamisch (d. h. bedarfsgerecht und abhängig von Spezifikationen des abzuarbeitenden Programms) aus dem Hauptthread heraus erzeugt und gestartet wird (Schritt S3b). Der Zusatzthread wird dabei auf einem zweiten logischen Prozessor ausgeführt. In dem Zusatzthread erfolgt nun – parallel zum (Weiter) Arbeiten des Hauptthreads (Schritt S3a) – die Abarbeitung einen weiteren Programms/Programmabschnitts, wie in Schritt S3c dargestellt ist, wodurch die Effizienz des Prozessors erhöht wird. Dabei können auch nur kurze Programmabschnitte in dem Hauptthread und dem Zusatzthread parallel bearbeitet werden, da durch das Erzeugen des Zusatzthreads aus dem Hauptthread heraus mittels des Prozessorbefehls (fork) ein sinnvolles Belegen der einzelnen Threads entsprechend den Programmanforderungen und damit ein effizientes uns zeitsparendes Abarbeiten von Programmabschnitten garantiert ist.

[0029] Beim Erzeugen des Zusatzthreads durch den Prozessorbefehl (fork) wird beispielsweise ein Zero-Flag im Hauptthread gelöscht und das Zero-Flag im Zusatzthread gesetzt, wodurch sich die beiden Threads voneinander unterscheiden. Zudem kann ein neues Flagbit in einem Register (zum Beispiel einem EFLAGS Register, in dem Zustandsinformationen abgelegt werden) vorgesehen sein, das beispielsweise dann gesetzt wird, wenn der Zusatzthread läuft.

[0030] Selbstverständlich kann als Unterscheidungsmerkmal auch ein Programmzähler, der für jeden Thread separat vorhanden ist, oder ein Register dienen. Ein Unterscheidungsmerkmal der beiden Threads ist wichtig, um beispielsweise bei einer Unterbrechungssaufforderung, welche bei der Beschreibung von Fig. 2 im Detail behandelt wird, zwischen den beiden Threads unterscheiden zu können bzw. einem Benutzerprogramm (beispielsweise einem Betriebssystem) anzeigen zu können, dass ein Zusatzthread läuft sowie um festlegen zu können, in welchem der Threads eine Unterbrechung behandelt werden soll, bzw. welcher Thread angehalten werden soll.

[0031] Für den auf dem ersten logischen Prozessor ablaufenden Hauptthread kann ein erster Registersatz vorgesehen sein und für den auf dem zweiten logischen Prozessor ablaufenden Zusatzthread ein zweiter Registersatz. Der Zusatzthread kann bei dessen Start die selben Registerwerte aufweisen, die der Hauptthread beim Aufrufen des (fork) Prozessorbefehls hat. Um die Initialisierung des Zusatzthreads möglichst wenig zeitaufwändig zu machen, können die entsprechenden Registerwerte (d. h. die Registerwerte des zweiten Registersatzes) beispielsweise in Phasen, in denen der Zusatzthread nicht läuft kontinuierlich den Registerwerten des Hauptthreads (d. h. den Registerwerten des ersten Registersatzes) angeglichen werden. Dies erspart ein zeitaufwändiges Laden der Registerwerte des zweiten Registersatzes beim Erzeugen/Starten des Zusatzthreads, wodurch der Zusatzthread ohne Zeitverzögerung mit der Abarbeitung eines Programms/Programmabschnitts beginnen kann (Schritt S3c), was das Verfahren noch effizienter macht. Auf diese Angleichung der Registerwerte des Zusatzthreads wird in Fig. 3 sowie Fig. 4 ausführlich eingegangen.

[0032] Der Prozessorbefehl (fork) kann – wie in diesem Ausführungsbeispiel dargestellt – dazu vorgesehen sein, den Zusatzthread sowohl zu erzeugen, als auch zu starten. Selbstverständlich sind für das Erzeugen und das Starten des Zusatzthreads auch zwei separate Prozessorbefehle vorstellbar. Zudem ist es möglich durch ein mehrfaches Aufrufen des Prozessorbefehls (fork) eine größere – vorbestimmte – Anzahl von Zusatzthreads zu erzeugen, die durch die Anzahl der logischen Prozessoren im System begrenzt ist.

[0033] Es können nie mehr Threads (Hauptthreads und Zusatzthreads) als logische Prozessoren vorhanden sein. Übersteigt die Anzahl der Aufrufe des Prozessorbefehls (fork), der jedes mal einen Zusatzthread erzeugt, die vorbestimmte Anzahl von Zusatzthreads (hier in diesem Ausführungsbeispiel genau einen Zusatzthread, bedingt durch die zwei logischen Prozessoren im System) so wird eine Unterbrechungsaufforderung erzeugt. Dadurch wird verhindert, dass die Zahl der Threads größer als die Zahl der logischen Prozessoren werden kann.

[0034] Das Erzeugen des bzw. der Zusatzthreads erfolgt dabei direkt aus dem Hauptthread heraus, als unmittelbare Reaktion auf den entsprechenden Befehl (fork) und somit entsprechend dem Bedarf des Benutzerprogramms. Dadurch wird auch die parallele Verarbeitung nur kurzer Programmabschnitte durch Haupt- und Zusatzthread sinnvoll. Aus Sicht eines Betriebssystems läuft nach dem Erzeugen und Starten des Zusatzthreads weiterhin nur ein Thread – der Hauptthread. Ein Beanspruchen des Zusatzthreads zum Bearbeiten von Aufgaben des Betriebssystems wird dadurch verhindert, die entsprechenden Ressourcen sind dem Benutzerprogramm vorbehalten.

[0035] In Schritt S3d wird – im Allgemeinen annähernd zeitgleich zu Schritt S3c – eine Variable run_Z auf einen Wert 1 gesetzt und dadurch ausgedrückt, dass ein Programm/Programmabschnitt in einem Zusatzthread bearbeitet wird. Durch das Festlegen dieser Variable und deren Speichern in einem Register (falls die Variable ein Flagbit ist) bzw. in einem Hauptspeicher des Prozessors ist beispielsweise durch den Hauptthread abrufbar, ob gerade ein Programmabschnitt in dem Zusatzthread abgearbeitet wird, oder ob der Zusatzthread mit der Abarbeitung fertig ist, was eine essentielle Information für eine effiziente Synchronisierung der wenigstens zwei Threads ist und auch bei einer Unterbrechungsbehandlung (siehe **Fig. 2**) eine wichtige Rolle spielen kann. Die Variable run_Z kann dabei beispielsweise dem (wie oben beschriebenen) neuen Flagbit (zum Beispiel im EFLAGS Register) entsprechen, oder auch eine weitere zusätzlich definierte Variable sein.

[0036] In Schritt S4 ist die Abarbeitung des Programms/Programmabschnitts im Hauptthread beendet. Der Hauptthread kann nun entweder, initiiert durch den Prozessor, mit der Abarbeitung eines neuen Programms/Programmabschnitts beginnen – was in diesem Ausführungsbeispiel nicht vorgesehen ist – oder der Hauptthread muss zum Beispiel aus Synchronisierungsgründen, beispielsweise wenn der Hauptthread zum Weiterarbeiten auf Ergebnisse des Zusatzthreads angewiesen ist, auf das Ende der Programmbearbeitung im Zusatzthread warten, bis er durch den Prozessor entweder zur Abarbeitung eines neuen Programms/Programmabschnitts vorgesehen oder aber beendet wird. Letzteres wird im folgenden näher ausgeführt.

[0037] Dafür kann in Schritt S5 eine Abfrage $run_Z = 0$? vorgesehen sein, mit der mittels der in Schritt S3d festgelegten Variablen run_Z überprüft wird, ob der Zusatzthread noch arbeitet. Ist run_Z ungleich 0, wird also noch ein Programm/Programmabschnitt im Zusatzthread verarbeitet, so ist in diesem Ausführungsbeispiel ein Prozessorbefehl (wait) vorgesehen (Schritt S6a), der den Hauptthread in eine Art Wartemodus versetzt (Schritt S6b). Aus diesem Wartemodus heraus erfolgt ein wiederholtes Abfragen der Variablen run_Z (Schritt S5), solange bis run_Z gleich 0 ist, der Zusatzthread also fertig ist. Während dieser Zeit werden keine Programme/Programmabschnitte im Hauptthread verarbeitet.

[0038] Der Prozessorbefehl (wait) kann natürlich ebenso an den Zusatzthread ergehen, so dass der Zusatzthread auf die Fertigstellung der Abarbeitung des Programms/Programmabschnitts im Hauptthread wartet. Selbstverständlich kann – wie bereits oben erwähnt – auch vorgesehen sein, dass im Hauptthread nach Abarbeitung des Programmabschnitts sofort ein neuer Programmabschnitt/neues Programm abgearbeitet wird, so dass sowohl der Prozessorbefehl (wait), als auch die Abfrage $run_Z = 0$? (Schritt S5) an dieser Stelle entfallen können.

[0039] Während des Wartemodus erfolgt die Weiterverarbeitung des Programms/Programmabschnitts im Zusatzthread (siehe Schritt S3b) bis ein Prozessorbefehl (stop) aufgerufen wird, wie in Schritt S7a dargestellt ist. Der Prozessorbefehl (stop) wird dabei beispielsweise aufgerufen, wenn das Programm/der Programmabschnitt im Zusatzthread fertig abgearbeitet ist. Der Prozessorbefehl (stop) kann jedoch auch zum Stoppen eines aktuellen Zusatzthreads vorgesehen sein, wenn bereits ein weiterer Zusatzthread läuft, zum Beispiel um ein Überschreiten der vorbestimmten Zahl von Zusatzthreads zu unterbinden. Dies ist sinnvoll, da die Anzahl der Threads (Hauptthreads und Zusatzthreads) nie größer als die Anzahl der logischen Prozessoren im System werden kann. Zudem kann ein Stoppen des Zusatzthreads mittels des Prozessorbefehls (stop) aus Effizienzgründen sinnvoll sein, da die Gesamtleistung des physikalischen Prozessors im Allgemeinen nicht linear mit der Anzahl der Threads ansteigt.

[0040] Durch den Aufruf des Prozessorbefehls (stop) – Schritt S7a – wird der Zusatzthread abhängig von den Spezifikationen des Programms beendet (Schritt S7b), und die Variable run_Z wird auf den Wert 0 gesetzt, wie in Schritt S7c dargestellt ist, wodurch der Zusatzthread nun auch für den Hauptthread sichtbar beendet ist.

[0041] Durch das Setzen der Variable run_Z auf den Wert 0 erhält der Hauptthread durch die Abfrage in Schritt S5 nun die Information, dass der Zuatzthread beendet ist und wird daraufhin ebenso beendet, wie in Schritt S8 dargestellt ist.

[0042] Im folgenden kann dem Hauptthread vom Prozessor wieder ein Programm/Programmabschnitt zur Abarbeitung zugewiesen werden und das dargestellte Verfahren schließt sich wieder an Schritt S1 an.

[0043] Mittels des beschriebenen Verfahrens können – mit Hilfe des erfindungsgemäßen Prozessors – Programme/Programmabschnitte effizient und zeitsparend parallelisiert werden, wobei die Zuweisung von abzuarbeitenden Programmen/Programmabschnitten an die Threads gemäß den Anforderungen und Spezifikationen des Benutzerprogramms erfolgt. So können Programme bedeutend beschleunigt werden, wodurch auch kurzzeitige Parallelisierungen sinnvoll sind. Durch das Vorsehen von Zusatzthreads als dynamisch und bedarfsgerecht – d. h. entsprechend einem Bedarf eines Benutzers bzw. abzuarbeitenden Programms – aus dem Hauptthread heraus erzeugbare Instanzen und der Verlegung von deren Erzeugung auf die Hardware Ebene – wobei

dabei auf ohnehin vorhandene Ressourcen des Prozessors zurückgegriffen wird –, wird der Durchsatz an Daten gesteigert und die Leistungsfähigkeit des Prozessors erhöht. Anders als beim gängigen Hyper-Threading, bei dem einzelne Threads in der Regel über Unterbrechungen miteinander kommunizieren können, ist in diesem Verfahren bzw. durch den erfindungsgemäßen Prozessor nun eine einfache, effektive und insbesondere schnelle Kommunikation der Threads untereinander über Register möglich.

[0044] Das Verfahren ist sowohl explizit – durch Einfügen der entsprechenden Prozessorbefehle in einen Programmcode eines Benutzerprogramms – als auch automatisch mittels einer Unterstützung durch einen Compiler durchführbar, wobei letzteres eine effiziente Ausnutzung des Prozessors auch ohne zusätzlichen Aufwand für einen Programmierer ermöglicht. Bei der automatischen Realisierung des Verfahrens muss ein Compilercode entsprechend den erfindungsgemäßen Prozessorbefehlen angepasst und optimiert werden.

[0045] Fig. 2 zeigt ein Ablaufdiagramm für ein Verfahren zur parallelen Verarbeitung von Programmen in zwei Threads mittels des erfindungsgemäßen Prozessors in einem weiteren Ausführungsbeispiel.

[0046] Die Schritte S1 bis einschließlich S3d sind dabei identisch mit den Schritten S1 bis S3d des in Fig. 1 vorgestellten Ausführungsbeispiels und werden daher an dieser Stelle nicht mehr im Detail beschrieben.

[0047] In Schritt S4 ergeht eine Unterbrechungsaufforderung an den Prozessor. Dabei gibt es zum einen so genannte synchrone Unterbrechungen, die durch ein aktuell ausgeführtes Programm selbst herbeigeführt werden. Synchrone Unterbrechungen lassen den Ablauf von in einer Befehlspipeline eines Threads abgespeicherten und auszuführenden Befehlen zur Abarbeitung eines Programms/Programmabschnitts in ihrer Reihenfolge unverändert. Eine Befehlspipeline ist dabei eine Aufreihung von Befehlen die nacheinander in dem Thread abgearbeitet werden. Synchrone Unterbrechungen sind beispielsweise fehlende Adressübersetzungsinformationenen (TLB (Translation Lookaside Buffer) – Miss). Des weiteren können asynchrone Unterbrechungen auftreten, die ohne Bezug zu einem aktuellen Programm sein können, wie beispielsweise Nachrichten von weiteren Prozessoren oder Peripheriegeräten, durch welche die in der Befehlspipeline des Threads abgespeicherten und auszuführenden Befehle verworfen werden müssen, um stattdessen Befehle einer Routine zur Behandlung der Unterbrechung laden und bearbeiten zu können.

[0048] Im folgenden soll die Behandlung einer asynchronen Unterbrechung erläutert werden, selbstverständlich kann das Verfahren aber auch bei synchronen Unterbrechungen durchgeführt werden.

[0049] In Schritt S5a wird der Hauptthread auf Grund der in Schritt S4 aufgetretenen Unterbrechungsaufforderung durch den Prozessor angehalten. In diesem Zusammenhang muss ein so genannter momentaner Kontext des Hauptthreads (Registerinhalte, Adresse der Unterbrechung) in einem Arbeitsspeicher abgelegt und dadurch gesichert werden, um das gerade ausgeführte Programm/Programmabschnitt nach der Unterbrechungsbehandlung an der selben Stelle fortsetzen zu können. Im Anschluss muss der Prozessor Daten zur Behandlung der Unterbrechung (beispielsweise die Adresse eines bestimmten Befehls) in Registern für die Unterbrechungsbehandlung ablegen, das heißt ein Kontext für die eigentliche Behandlung der Unterbrechung muss geladen werden. Die sich bei Auftreten einer Unterbrechungsaufforderung in der Befehlspipeline des Hauptthreads befindenden Befehle zur Abarbeitung des (unterbrochenen) Programms/Programmsbschnitts werden verworfen. Die Pipeline wird daraufhin mit für die Behandlung der Unterbrechung nötigen Befehlen gefüllt.

[0050] In diesem Ausführungsbeispiel nicht gezeigt, jedoch dennoch an dieser Stelle vorstellbar ist eine mögliche Abfrage eines Benutzerprogramms (beispielsweise eines Betriebssystems), die beim Auftreten einer Unterbrechungsaufforderung vorgesehen sein kann um festzustellen, ob ein Zusatzthread läuft. Dabei kann ein Kontrollregister vorgesehen sein, mit dem dem Benutzerprogramm angezeigt wird, dass der Zusatzthread läuft. Dieses Kontrollregister kann den beim Anlaufen des Zusatzthreads gesetzen Zero-Flag – beispielsweise die weiter oben beschriebene Variable run_Z – aufweisen. Existiert wie in diesem Ausführungsbeispiel ein Zusatzthread, so können sich die im folgenden beschriebenen Verfahrensschritte an eine derartige Abfrage anschließen. Gibt es keinen Zusatzthread wird die Unterbrechungsbehandlung nur auf den Hauptthread bezogen.

[0051] Parallel zum Anhalten des Hauptthreads gibt das Benutzerprogramm (in diesem Beispiel im folgenden das Betriebssystem) einen Prozessorbefehl (save) ab (Schritt S5b), durch den der Zusatzthread angehalten und dessen Registerwerte gesichert werden (Schritt S5c). Der Zusatzthread befindet sich nun in einer Art Stand-by Modus und wartet auf weitere Befehle des Betriebssystems.

[0052] In Schritt S6 erfolgt die Behandlung der Unterbrechung, d. h. die Ausführung einer Unterbrechungsroutine zur Bearbeitung der Unterbrechung, in dem Hauptthread. Selbstverständlich kann die Unterbrechungsroutine aber auch im Zusatzthread ausgeführt werden und der Hauptthread mit Hilfe des Prozessorbefehls (save) in den Stand-by Modus gesetzt

werden.

[0053] Ebenso vorstellbar ist es – beispielsweise bei einer synchronen Unterbrechung, welche keine komplexe Behandlung im Betriebssytem erfordert – dass der Zusatzthread überhaupt nicht angehalten wird und der Prozessorbefehl (save) demnach an dieser Stelle entfallen kann, der Zusatzthread also von der Unterbrechung bzw. Unterbrechungsbehandlung unberührt mit der Abarbeitung des Programms/Programabschnitts fortfahren kann.

[0054] Der Prozessorbefehl (save), der in diesem Ausführungsbeispiel von dem Betriebssystem bei der Unterbrechungsbehandlung abgegeben wird, kann ebenso von jedem anderen Benutzerprogramm angewandt werden, zum Beispiel wenn der Zusatzthread abgebrochen werden soll.

[0055] In Schritt S7 ist im folgenden eine Abfrage vorgesehen, ob die Unterbrechungsbehandlung abgeschlossen ist. Ist dies nicht der Fall, wird die Unterbrechungsbehandlung im Hauptthread fortgesetzt (Schritt S6).

[0056] Ist die Unterbrechungsbehandlung beendet, muss die Befehlspipeline des Hauptthreads erneut geleert und neu gefüllt werden, um das vorher unterbrochene Programm/Programmabschnitt fortsetzen zu können. Danach setzt der Hauptthread das Abarbeiten des vorher unterbrochenen Programms/Programmabschnitts fort (Schritt S8a). Gleichzeitig wird durch das Betriebssystem ein Prozessorbefehl (restore) abgegeben (Schritt S8b), durch den der Zusatzthread mit einem vorgegebenen Registerwert wieder gestartet wird. Die Abarbeitung des vorher durch den Prozessorbefehl (save) unterbrochenen im Zusatzthread bearbeiteten Programms/Programmabschnitts wird im Zusatzthread fortgesetzt, wie in Schritt S8c dargestellt ist.

[0057] Zusätzlich für das Wiederstarten des Zusatzthreads nach einer Unterbrechungsbehandlung kann der Prozessorbefehl (restore) auch von jedem weiteren Benutzerprogramm zum erneuten Starten des Zusatzthreads verwendet werden.

[0058] Die Abarbeitung des Programms/Programmabschnitts im Hauptthread ist in Schritt S10 beendet. Im Gegensatz zu Fig. 1 ist an dieser Stelle kein Prozessorbefehl (wait) und auch keine Abfrage, ob der Zusatzthread noch läuft, vorgesehen, sondern der Hauptthread wird hier automatisch beendet, wenn die Abarbeitung des Programmabschnitts im Hauptthread abgeschlossen ist. Dies kann beispielsweise der Fall sein, wenn Hauptthread und Zusatzthread völlig unabhängige Programmabschnitte zur Abarbeitung zugewiesen wurden und der Hauptthread demnach auch nicht auf Ergebnisse des Zusatzthreads warten muss.

[0059] Nach Beenden der Programmbearbeitung im Hauptthread kann dem Hauptthread vom Prozessor beispielsweise ein neues Programm/einer neuer Programmabschnitt zur Abarbeitung zugewiesen werden.

[0060] In Schritt S11a ergeht der Prozessorbefehl (stop), der den Zusatzthread – beispielsweise nach Fertigstellen der Abarbeitung des Programms im Zusatzthread – beendet (Schritt S11b).

[0061] In Schritt S11c wird die Variable run_Z daraufhin wieder auf den Wert 0 gesetzt, wodurch signalisiert wird, dass der Zusatzthread nicht mehr läuft uns dass nun ein neuer Zusatzthread gestartet werden kann. Ebenso kann die Variable run_Z natürlich dazu verwendet werden, um dem Hauptthread bei einer entsprechenden Abfrage anzuzeigen, dass der Zusatzthread fertig ist, wie es bei der Beschreibung von Fig. 1 ausgeführt ist.

[0062] Wie bereits bei der Beschreibung von Fig. 1 erwähnt, sind auch bei diesem Ausführungsbeispiel mehrere durch den Prozessorbefehl (fork) erzeugbare Zusatzthreads vorstellbar – abhängig von den Spezifikationen des Systems (Anzahl logischer Prozessoren) sowie von den Voreinstellungen und Anforderungen eines Benutzerprogramms (und damit den Bedürfnissen des Benutzers). Damit kann das Verfahren optimal an die Vorstellungen des Benutzers und die Ansprüche der abzuarbeitenden Programme angepasst und die Effizienz des Prozessors entsprechend gesteigert werden, auch bzw. gerade wenn nur kurze Programmabschnitte in den Threads parallel verarbeitet werden sollen.

[0063] Fig. 3 zeigt ein Ablaufdiagramm für einen Teilbereich des in Fig. 1 dargestellten Verfahrens. Der dargestellte Teilbereich beschreibt dabei ein Initialisieren von Registern und ein Setzen von Registerwerten für den Zusatzthread sowie ein Angleichen der Register des Zusatzthreads an die entsprechenden Register des Hauptthreads, für den Fall dass der Zusatzthread nicht läuft.

[0064] Schritt S1 beschreibt eine – in Fig. 1 nicht angesprochene – Initialisierung des erfindungsgemäßen Prozessors, beispielsweise bei einem Anschalten oder auch nach einem Reset.

[0065] In Schritt S2 werden die Register des Hauptthreads (d. h. die Register des ersten Registersatzes) sowie des Zusatzthreads (d. h. die Register des zweiten Registersatzes) mit den – in diesem Ausführungsbeispiel – selben Werten initialisiert. Natürlich ist auch eine Initialisierung mit unterschiedlichen Werten möglich, beispielsweise wenn ein Benutzer vorsieht, dass der Zusatzthread ohnehin ein komplett anderes Programm abarbeiten und demnach mit anderen Registerwerten anlaufen soll, als der Hauptthread.

[0066] Für jedes Register des Zusatztthreads (bzw. auch für Gruppen von Registern des Zusatzthreads) können beispielsweise – wie in Fig. 3 auf der rechten Seite gezeigt – zwei Zustandsbits (Valid, Same) vorgesehen sein. Diese Zustandsbits beschreiben einen (Initialisierungs-)status des entsprechenden Registers des Zusatztthreads. Das Zustandsbit (Valid) zeigt dabei an, ob ein Register des Zusatzthreads einen gültigen Inhalt hat (Valid = 1), das andere Zustandsbit (Same) beschreibt, ob das entsprechende Register des Hauptthreads denselben Wert enthält, wie das Register des Zusatzthreads (Same = 1). Bei der Initialisierung des Prozessors werden die entsprechenden Zustandsbits (Valid, Same) auf den Wert 1 gesetzt, wie in einem Zustandsdiagramm auf der rechten Seite von Fig. 3 dargestellt ist. Die Register des Zusatzthreads weisen demnach in diesem Ausführungsbeispiel nach der Initialisierung die selben (und gültigen) Inhalte wie die Register des Hauptthreads auf.

[0067] In Schritt S3 beginnt der Hauptthread – initiiert durch den Prozessor – mit der Abarbeitung eines Programms/Programmabschnitts.

[0068] In Schritt S4 ändert sich ein Inhalt eines Registers des Hauptthreads. Da die Register des Hauptthreads und des Zusatzthreads jedoch bei einem Erzeugen / Starten des Zusatzthreads vorteilhafterweise die selben Inhalte aufweisen sollten, ist in Schritt S5 ein Kopieren des Inhalts des veränderten Registers des Hauptthreads in das entsprechende Register des Zusatzthreads vorgesehen, d. h. es ist ein permanentes Nachführen der Registerwerte des Zusatzthreads unmittelbar nach dem Ändern der Registerwerte des Hauptthreads vorgesehen.

[0069] Das betreffende Register von Haupt- und Zusatzthread ist nach diesem Kopieren wieder gleich und die Zustandsbits (Valid, Same) sind weiterhin auf den Wert 1 gesetzt, um auszudrücken, dass das entsprechende Register des Zusatzthreads einen gültigen Inhalt besitzt und die Register von Zusatzthread und Hauptthread gleich sind.

[0070] In Schritt S6 wird aus dem Hauptthread heraus der Prozessorbefehl (fork) aufgerufen und der Zusatzthread wird erzeugt bzw. gestartet (Schritt S7b), wie bereits bei Fig. 1 beschrieben wurde. Der Hauptthread fährt derweil mit der Abarbeitung des Programms/Programmabschnitts fort, wie in Schritt S7a gezeigt ist.

[0071] Das sofortige Initialisieren der Register des Zusatzthreads und das Setzen der Registerwerte des Zusatzthreads auf die selben Werte wie die entsprechenden Register des Hauptthreads bei der Initialisierung des Prozessors sowie das permanente Angleichen der Registerwerte des Zusatzthreads an die Registerwerte des Hauptthreads während der Zusatzthread nicht läuft, als noch vor dem Erzeugen/Starten des Zusatzthreads, und zwar in dem Moment in dem das entsprechende Register des Hauptthreads modifiziert wird, erlaubt ein sofortiges Anlaufen des Zusatzthreads nach Abgeben des Prozessorbefehls (fork) mit den passenden Registerwerten. Dadurch können zeit- und ressourcenintensive Lade- bzw. Initialisierungsprozesse für die Register des Zusatzthreads beim Abgeben des (fork) Prozessorbefehls gespart werden und das erfindungsgemäße Verfahren weiter optimiert werden.

[0072] Dadurch dass die Registerwerte des Zusatzthreads bei der Initialisierung auch auf andere Werte, als die Registerwerte des Hauptthreads gesetzt werden können, beispielsweise wenn ein Benutzer vorsieht, dass Haupt- und Zusatzthread ohnehin mit komplett anderen Werten gestartet werden sollen, kann in diesem Fall auch ein Angleichen der Registerwerte des Zusatzthreads an die Registerwerte des Hauptthreads – während der Zusatzthread nicht läuft – entfallen. Das Verfahren ist folglich auch beim Initialisieren bzw. Setzen der Registerwerte des Zusatzthreads komplett den Vorstellungen des Benutzers sowie den Anforderungen der abzuarbeitenden Programme anpassbar.

[0073] Fig. 4 zeigt ein Ablaufdiagramm für einen weiteren Teilbereich des in Fig. 1 dargestellten Verfahrens. Der hier dargestellte Teilbereich beschreibt ein Verändern bzw. Setzen von Registerwerten bzw. der Zustandsbits (Valid, Same) (siehe Fig. 3) für den Zusatzthread, für den Fall dass der Zusatzthread läuft, d. h. nach dem Abgeben des Prozessorbefehls (fork), und schließt sich direkt an den in Fig. 3 gezeigten Teilbereich des erfindungsgemäßen Verfahrens an.

[0074] Schritt S7a (Weiterarbeiten des Hauptthreads) sowie Schritt S7b (Erzeugen/Start des Zusatzthreads) entsprechen dabei den Schritten S7a und S7b aus Fig. 3.

[0075] Bei Weiterarbeiten des Hauptthreads (Schritt S7a) wird ein Register des Hauptthreads verändert, wie in Schritt S8 dargestellt ist. Ist das Zustandsbit (Valid) des entsprechenden Registers des Zusatzthreads auf den Wert 1 gesetzt, wie in diesem Ausführungsbeispiel vorgesehen, dann wird das Zustandsbit (Same) des Registers auf den Wert 0 gesetzt, wie in einem Zustandsdiagramm auf der rechten Seite von Fig. 4 gezeigt wird. Die betreffenden Register von Hauptthread und Zusatzthread unterscheiden sich jetzt potentiell (Same = 0). Das Zustandsbit (Valid) kann dabei weiterhin auf dem Wert 1 bleiben, der Zusatzthread arbeitet also mit unverändertem Register weiter. Ein Angleichen des Registers des Zusatzthreads an das entsprechende Register des Hauptthreads (und als Konsequenz das Erreichen eines Zustands mit Zustandsbit (Same = 1)) ist

bei dieser Ausführung der Erfindung nicht vorgesehen, solange der Zusatzthread läuft und das Zustandsbit (Valid) gleich 1 ist.

[0076] Für den Fall dass ein Register im Haupt-Thread verändert wird und das Zustandsbit (Valid) des entsprechenden Registers des Zusatzthreads auf den Wert 0 gesetzt ist, das entsprechende Register des Zusatzthreads also keinen gültigen Inhalt besitzt, dann kann der alte Inhalt des Registers des Hauptthreads in das entsprechende Register des Zusatzthreads kopiert werden, und das Zustandsbit (Valid) auf 1 sowie das Zustandsbit (Same) auf 0 gesetzt werden, um auszudrücken, dass sich das Register des Zusatzthreads von dem entsprechenden Register des Hauptthreads unterscheidet, das Register des Zusatzthreads dennoch aber einen gültigen Wert besitzt (Valid = 1). Diese Variante des Verfahrens ist in Fig. 4 jedoch nicht dargestellt.

[0077] Nicht nur die Register im Hauptthread können sich bei Abarbeitung eines Programms/Programmabschnitts ändern, sondern selbstverständlich können sich beim Arbeiten des Zusatzthreads auch dessen Register ändern. Wird ein Register im Zusatzthread verändert (Schritt S9), so werden das Zustandsbit (Valid) des Registers auf den Wert 1 und das Zustandsbit (Same) auf den Wert 0 gesetzt. In diesem konkreten Fall ändert sich dabei an den Werten der Zustandsbits (Valid, Same) nichts, da das Zustandsbit (Same) auf Grund des Änderns des Registers des Hauptthreads (Schritt S8) bereits auf den Wert 0 gesetzt worden ist.

[0078] Der Zusatzthread arbeitet mit dem veränderten Register weiter. Bei einem Lesen und gleichzeitigem Schreiben der Register im Zusatzthread ist bei dieser Ausführung keine Angleichung an die entsprechenden Register des Hauptthreads vorgesehen.

[0079] Wird ein Register im Zusatzthread jedoch beispielsweise nur gelesen und nicht geschrieben, und ist das Zustandsbit (Valid) auf 0 gesetzt, so wird der Inhalt des entsprechenden Registers des Hauptthreads in das Register des Zusatzthreads kopiert, und die beiden Zustandsbits (Valid, Same) werden auf den Wert 1 gesetzt. Der Zusatzthread muß also, wenn er lesend verwendet wird, in jedem Fall einen gültigen Inhalt besitzen (Valid = 1).

[0080] Ein Setzen des Zustandsbits (Valid) auf den Wert 0 und ein gleichzeitiges Setzen des Zustandsbits (Same) auf den Wert 1 kann jedoch in keinem Fall vorkommen, da der Hauptthread in jedem Fall gültige Registerinhalte aufweist.

[0081] In Schritt S10a ergeht der Prozessorbefehls (stop) und der Zusatzthread wird angehalten und beendet (Schritt S10b). Nach Anhalten des Zusatzthreads (dies kann beispielsweise auch mit dem Prozessorbefehl (save) geschehen) werden seine Registerinhalte nicht mehr benötigt. Daraufhin können, wie auf der rechten Seite von Fig. 4 dargestellt ist, die Zustandsbits (Valid) der Register des Zusatzthreads auf den Inhalt der entsprechenden Zustandsbits (Same) gesetzt werden, in diesem konkreten Fall wird das Zustandsbit (Valid) also auf den Wert 0 gesetzt. Demnach sind in diesem Ausführungsbeispiel nach Anhalten des Zusatzthreads nur diejenigen Register des Zusatzthreads als gültig gekennzeichnet, die in beiden Threads (Hauptthread und Zusatzthread) dieselben Inhalte hatten, bei denen also vor Ergehen des (stop) Prozessorbefehls das Zustandsbit (Same) den Wert 1 hatte.

[0082] In Schritt S11 ist die Abarbeitung des Programms/Programmabschnitts im Hauptthread beendet, und im Hauptthread kann – hier wiederum unabhängig vom Zusatzthread – mit der Abarbeitung eines neuen Programms/Programmabschnitts begonnen werden.

Patentansprüche

1. Prozessor, der zur parallelen Ausführung von wenigstens zwei Threads mittels Hyper-Threading auf wenigstens zwei logischen Prozessoren geeignet ist,
dadurch gekennzeichnet, dass
– ein Prozessorbefehl (fork) vorgesehen ist, der dazu geeignet ist einen zweiten Thread zu erzeugen und/oder zu starten,
– ein Prozessorbefehl (stop) vorgesehen ist, der dazu geeignet ist einen aktuellen zweiten Thread zu beenden.

2. Prozessor nach Anspruch 1, dadurch gekennzeichnet, dass ein Prozessorbefehl (wait) vorgesehen ist, der dazu geeignet ist, dass ein erster Thread wartet, während der zweite Thread noch arbeitet, bzw. der zweite Thread wartet, während der erste Thread noch arbeitet.

3. Prozessor nach einem der Ansprüche 1 oder 2,
dadurch gekennzeichnet, dass
– ein Prozessorbefehl (save) vorgesehen ist, der dazu geeignet ist, den zweiten Thread anzuhalten und ein Register des zweiten Threads zu speichern,
– ein Prozessorbefehl (restore) vorgesehen ist, der dazu geeignet ist, den mittels des Prozessorbefehls (save) angehaltenen zweiten Thread mit einem vorgegebenen Registerinhalt wieder zu starten.

4. Prozessor nach einem der Ansprüche 1 bis 3, dadurch gekennzeichnet, dass eine Unterbrechungsaufforderung erzeugt wird, wenn die Anzahl der Prozessorbefehle (fork) eine vorbestimmte Anzahl von zweiten Threads übersteigt.

5. Prozessor nach einem der Ansprüche 1 bis 4, dadurch gekennzeichnet, dass ein erster Registersatz für den auf einem ersten logischen Prozessor ablaufenden ersten Thread sowie ein zweiter Registersatz für den auf einem zweiten logischen Prozessor ablaufenden zweiten Thread vorgesehen ist.

6. Prozessor nach Anspruch 5, dadurch gekennzeichnet, dass eine Zuordnung von zwei Zustandsbits (Valid, Same) für jedes Register des zweiten Registersatzes vorgesehen ist, die dazu eingerichtet sind anzuzeigen, ob ein Register des zweiten Registersatzes einen gültigen Inhalt hat (Valid = 1) und ob ein Register des ersten Registersatzes den selben Wert enthält, wie das entsprechende Register des zweiten Registersatzes (Same = 1).

7. Prozessor nach einem der Ansprüche 5 oder 6, dadurch gekennzeichnet, dass bei einer Initialisierung des Prozessors eine Initialisierung des ersten Registersatzes und des zweiten Registersatzes mit denselben Werten vorgesehen ist.

8. Prozessor nach Anspruch 7, dadurch gekennzeichnet, dass bei der Initialisierung des Prozessors ein Setzen der zwei Zustandsbits (Valid, Same) jedes Registers des zweiten Registersatzes auf den Wert 1 vorgesehen ist.

9. Prozessor nach einem der Ansprüche 5 bis 8, dadurch gekennzeichnet, dass eine Übereinstimmung des zweiten Registersatzes mit dem ersten Registersatz beim Abgeben des (fork) Prozessorbefehls vorgesehen ist.

10. Prozessor nach einem der Ansprüche 5 bis 9, dadurch gekennzeichnet, dass während der zweite Thread nicht läuft, ein kontinuierliches Setzen eines Registers des zweiten Registersatzes auf den selben Wert, wie das entsprechende Register des ersten Registersatzes vorgesehen ist.

11. Prozessor nach einem der Ansprüche 6 bis 10, dadurch gekennzeichnet, dass während der zweite Thread mit einem gültigen Register (Valid = 1) läuft und ein entsprechendes Register des ersten Registersatzes verändert wird, ein Setzen des Zustandsbits (Same) des Registers des zweiten Registersatzes auf den Wert 0 vorgesehen ist.

12. Prozessor nach einem der Ansprüche 6 bis 11, dadurch gekennzeichnet, dass während der zweite Thread mit einem Register mit Zustandsbit (Valid = 0) läuft und ein entsprechendes Register des ersten Registersatzes verändert wird, ein Kopieren des alten Inhalts des Registers des ersten Registersatzes in das entsprechende Register des zweiten Registersatzes vorgesehen ist sowie ein Setzen des Zustandsbits (Valid) auf den Wert 1 und ein Setzen des Zustandsbits (Same) auf den Wert 0 vorgesehen ist.

13. Prozessor nach einem der Ansprüche 6 bis 12, dadurch gekennzeichnet, dass bei einem Verändern eines Registers des zweiten Registersatzes ein Setzen des Zustandsbits (Valid) des Registers auf den Wert 1 und ein Setzen des Zustandsbits (Same) des Registers auf den Wert 0 vorgesehen ist.

14. Prozessor nach einem der Ansprüche 6 bis 13, dadurch gekennzeichnet, dass bei einem Lesen eines Registers des zweiten Registersatzes mit dem Zustandsbit (Valid = 0) ein Kopieren des Inhalts des entsprechenden Registers des ersten Registersatzes in das Register des zweiten Registersatzes vorgesehen ist und ein Setzen der Zustandsbits (Valid, Same) des Registers des zweiten Registersatzes auf den Wert 1 vorgesehen ist.

15. Prozessor nach einem der Ansprüche 6 bis 14, dadurch gekennzeichnet, dass bei einem Anhalten des zweiten Threads mittels des Prozessorbefehls (stop, save) ein Setzen der Zustandsbits (Valid) der Register des zweiten Registersatzes auf die Inhalte der entsprechenden Zustandsbits (Same) der Register des zweiten Registersatzes vorgesehen ist.

16. Prozessor nach Anspruch 15, dadurch gekennzeichnet, dass eine Kennzeichnung derjenigen Register des zweiten Registersatzes als gültig vorgesehen ist, die in dem ersten Registersatz und dem zweiten Registersatz denselben Inhalt hatten.

17. Prozessor nach einem der Ansprüche 1 bis 16, dadurch gekennzeichnet, dass ein neues Flagbit in einem Register vorgesehen ist, das gesetzt wird, wenn ein zweiter Thread läuft.

18. Verfahren zur parallelen Verarbeitung von Programmen durch den Prozessor aus einem der Ansprüche 1 bis 17, gekennzeichnet durch die folgenden Schritte:
– Definition eines auf einem ersten logischen Prozessor ablaufenden ersten Threads durch den Prozessor zur Abarbeitung eines Programms,
– Starten des ersten Threads durch den Prozessor,
– Erzeugen und Starten eines auf einem zweiten logischen Prozessor ablaufenden zweiten Threads durch den ersten Thread bedingt durch den Prozessorbefehl (fork),
– Beenden des zweiten Threads mittels des Prozessorbefehls (stop),
– Beenden des ersten Threads durch den Prozessor und/oder Starten der Abarbeitung eines neuen Programms in dem ersten Thread durch den Prozessor.

19. Verfahren nach Anspruch 18, dadurch gekennzeichnet, dass der erste Thread mit Hilfe des Prozessorbefehls (wait) wartet, bis der zweite Thread beendet ist und/oder der zweite Thread mit Hilfe des Prozessorbefehls (wait) wartet bis der erste Thread beendet ist.

20. Verfahren nach Anspruch 18 oder 19, dadurch gekennzeichnet, dass der zweite Thread mit denselben Registerwerten anläuft, die der erste Thread beim Abgeben des Prozessorbefehls (fork) aufweist.

21. Verfahren nach Anspruch 20, dadurch gekennzeichnet, dass während der zweite Thread nicht läuft, die Registerwerte eines zweiten dem zweiten Thread zugeordneten Registersatzes kontinuierlich auf dem selben Stand gehalten werden, wie die Registerwerte eines ersten dem ersten Thread zugeordneten Registersatzes.

22. Verfahren nach Anspruch 20 oder 21, dadurch gekennzeichnet, dass jedes Register des zweiten Registersatzes zwei Zustandsbits (Valid, Same) aufweist, die bei einem Aufrufen des Prozessorbefehls (fork) auf den Wert 1 gesetzt sind.

23. Verfahren nach Anspruch 21 oder 22, dadurch gekennzeichnet, dass während der zweite Thread nicht läuft, ein Register des zweiten Registersatzes auf den selben Wert, wie das entsprechende Register des ersten Registersatzes (Valid = 1, Same = 1) gesetzt wird.

24. Verfahren nach Anspruch 22, dadurch gekennzeichnet, dass während der zweite Thread mit einem gültigen Register (Valid = 1) läuft und ein entsprechendes Register des ersten Registersatzes verändert wird, das Zustandsbit (Same) des Registers des zweiten Registersatzes auf den Wert 0 gesetzt wird.

25. Verfahren nach Anspruch 22, dadurch gekennzeichnet, dass während der zweite Thread mit einem Register mit Zustandsbit (Valid = 0) läuft und ein entsprechendes Register des ersten Registersatzes verändert wird, der Inhalt des Registers des ersten Registersatzes in das entsprechende Register des zweiten Registersatzes kopiert wird und das Zustandsbit (Valid) des Registers des zweiten Registersatzes auf den Wert 1 sowie das Zustandsbit (Same) auf den Wert 0 gesetzt werden.

26. Verfahren nach Anspruch 22, dadurch gekennzeichnet, dass bei einem Verändern eines Registers des zweiten Registersatzes das Zustandsbit (Valid) des Registers auf den Wert 1 und das Zustandsbits (Same) auf den Wert 0 gesetzt werden.

27. Verfahren nach Anspruch 22, dadurch gekennzeichnet, dass bei einem Lesen eines Registers des zweiten Registersatzes mit dem Zustandsbit (Valid = 0) ein Inhalt des entsprechenden Registers des ersten Registersatzes in das Register des zweiten Registersatzes kopiert wird und die Zustandsbits (Valid, Same) des Registers des zweiten Registersatzes auf den Wert 1 gesetzt werden.

28. Verfahren nach einem der Ansprüche 22 bis 27, dadurch gekennzeichnet, dass bei einem Anhalten des zweiten Threads mittels des Prozessorbefehls (stop) die Zustandsbits (Valid) der Register des zweiten Registersatzes auf die Inhalte der entsprechenden Zustandsbits (Same) der Register des zweiten Registersatzes gesetzt werden.

29. Verfahren nach einem der Ansprüche 18 bis 28, dadurch gekennzeichnet, dass der erste Thread und der zweite Thread sich lediglich durch ein Flag oder durch einen Programmzähler unterscheiden.

30. Verfahren nach einem der Ansprüche 18 bis 29, dadurch gekennzeichnet, dass beim Auftreten einer Unterbrechungsaufforderung eine Abfrage durch ein Programm vorgesehen ist, ob ein zweiter Thread vorhanden ist.

31. Verfahren nach Anspruch 30, dadurch gekennzeichnet, dass ein Kontrollregister vorgesehen ist und bei der Unterbrechungsaufforderung dem Programm in dem Kontrollregister angezeigt wird, dass der zweite Thread läuft.

32. Verfahren nach Anspruch 30 oder 31, dadurch gekennzeichnet, dass bei der Unterbrechungsaufforderung der zweite Thread durch das Programm mittels des Prozessorbefehls (save) angehalten wird und dessen Registerwerte gesichert werden.

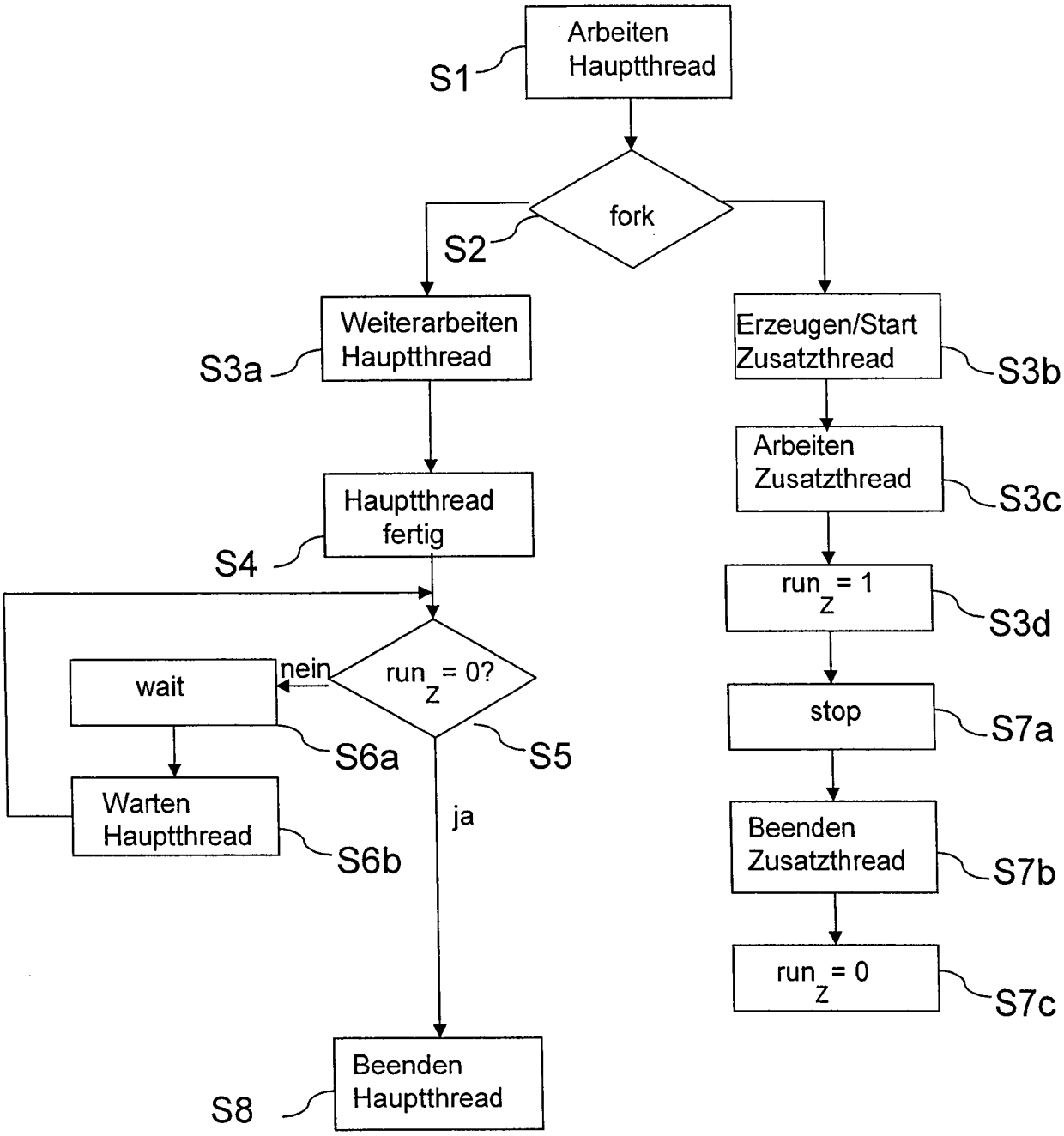
33. Verfahren nach Anspruch 32, dadurch gekennzeichnet, dass nach der Unterbrechungsbehandlung der zweite Thread mit einem vorgegebenen Registerwert durch das Programm mittels des Prozessorbefehls (restore) wieder gestartet wird.

34. Verfahren nach einem der Ansprüche 18 bis 33, dadurch gekennzeichnet, dass eine Unterbrechungsaufforderung erzeugt wird, falls die Anzahl der aufgerufenen Prozessorbefehle (fork) größer als eine vorbestimmte Anzahl an zweiten Threads ist.

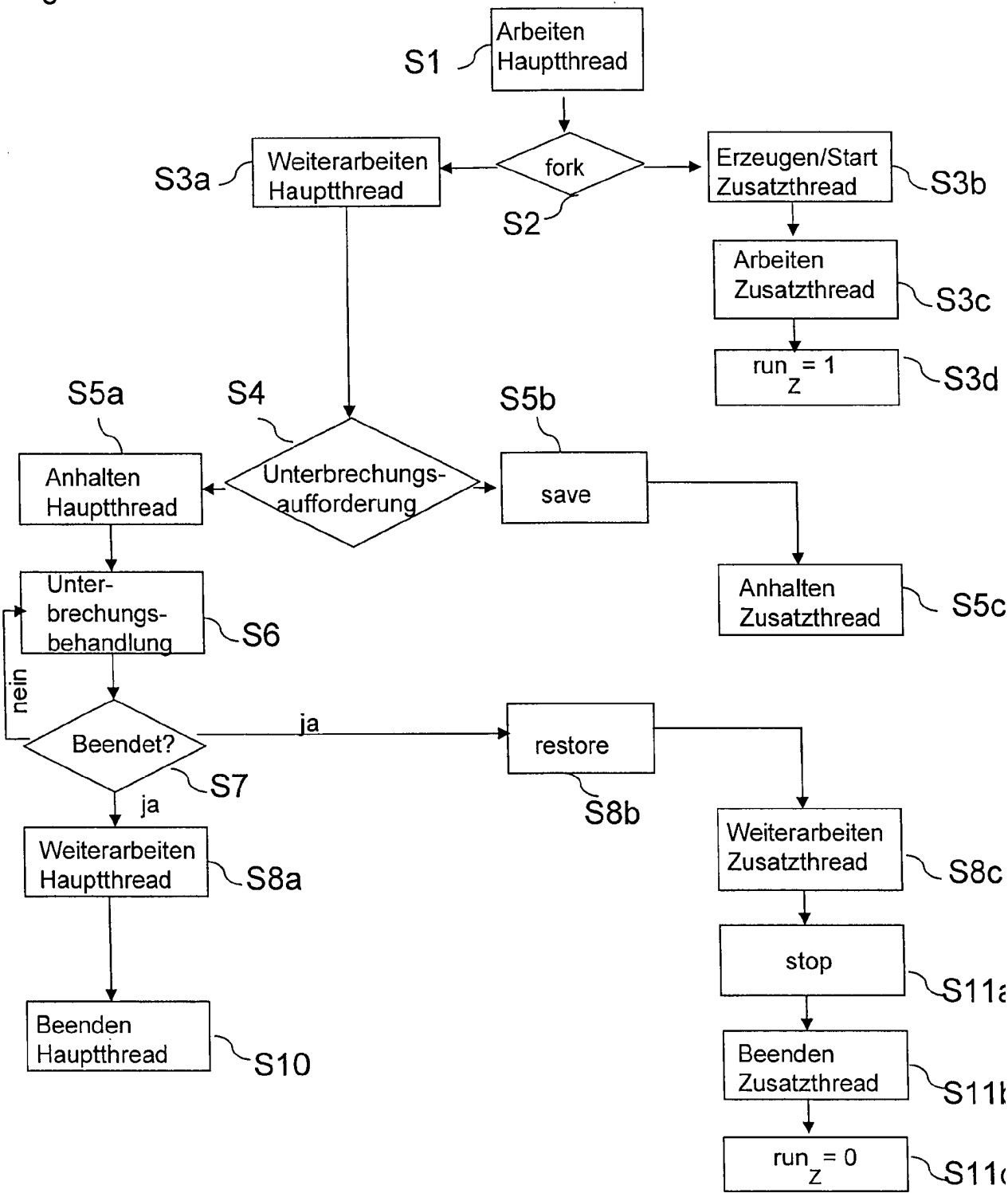
Es folgen 4 Blatt Zeichnungen

Anhängende Zeichnungen

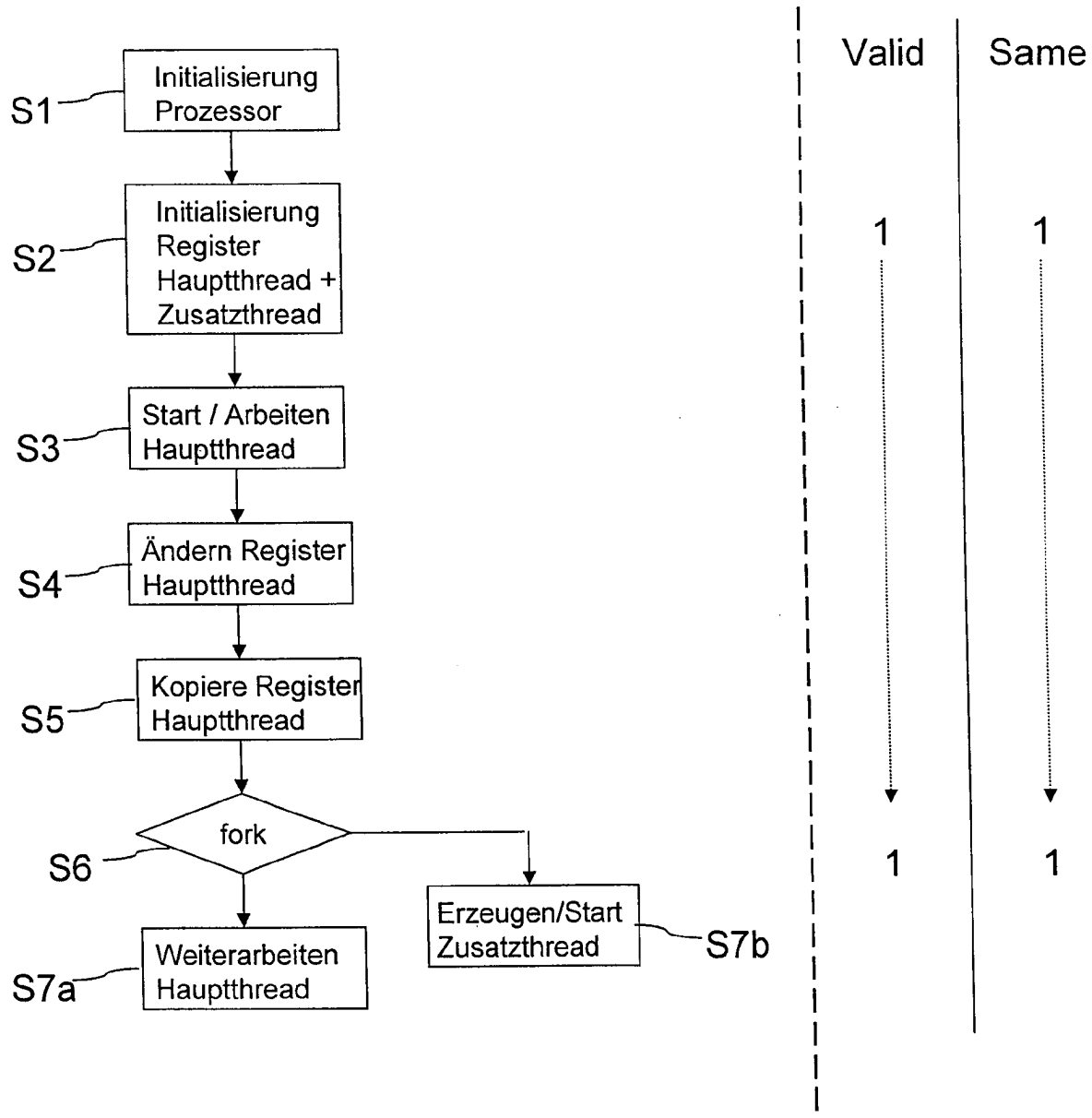
Figur 1



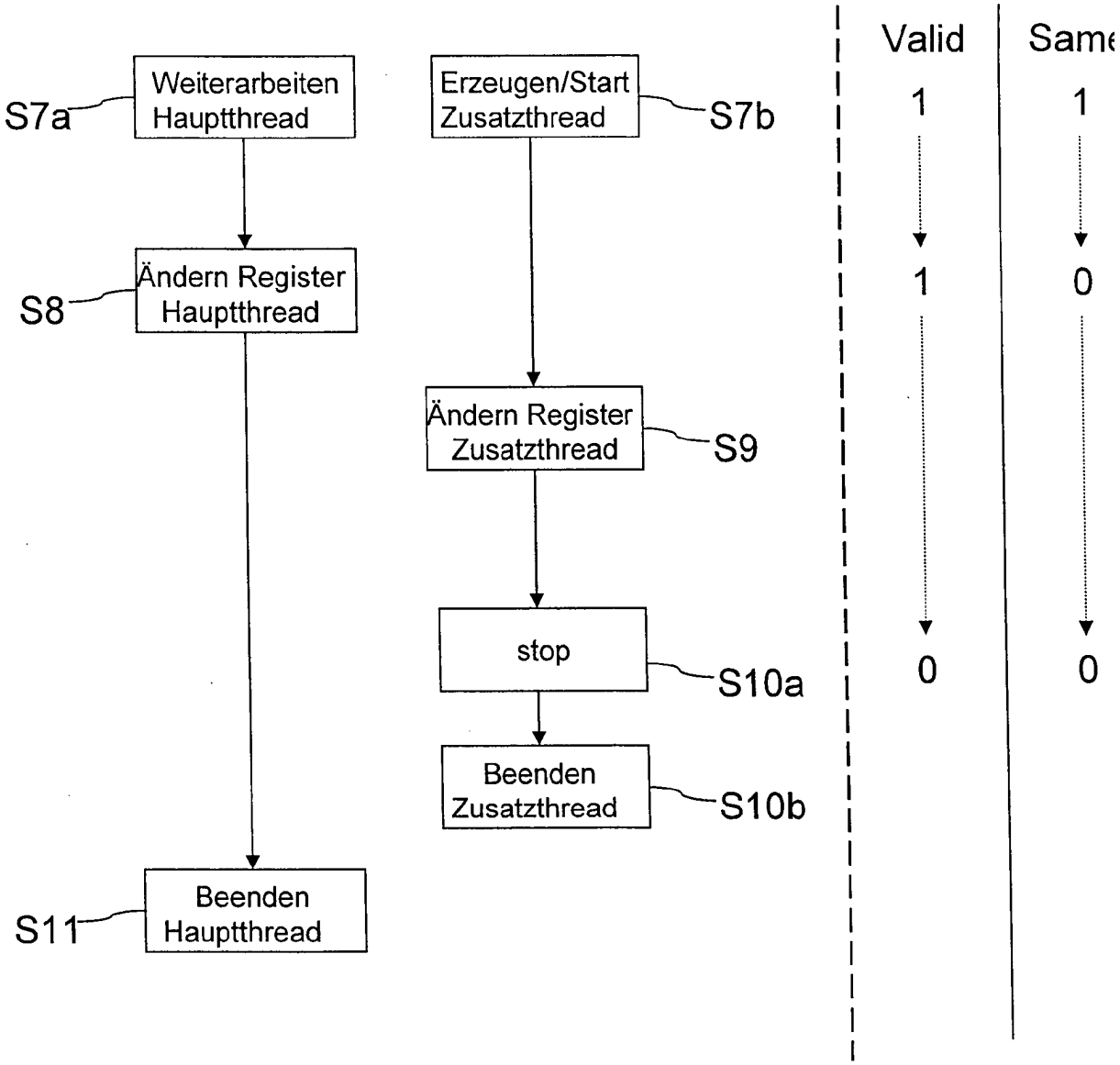
Figur 2


Arbeiten Hauptthread
S1
fork
S2
Weiterarbeiten Hauptthread
S3a
Erzeugen/Start Zusatzthread
S3b
Arbeiten Zusatzthread
S3c
run_z = 1
S3d
S4
Unterbrechungs-aufforderung
S5a
Anhalten Hauptthread
S5b
save
Anhalten Zusatzthread
S5c
Unter-brechungs-behandlung
S6
nein
Beendet?
S7
ja
ja
restore
S8b
Weiterarbeiten Hauptthread
S8a
Weiterarbeiten Zusatzthread
S8c
stop
S11a
Beenden Hauptthread
S10
Beenden Zusatzthread
S11b
run_z = 0
S11c

Figur 3


Valid
Same
S1
Initialisierung Prozessor
S2
Initialisierung Register Hauptthread + Zusatzthread
S3
Start / Arbeiten Hauptthread
S4
Ändern Register Hauptthread
S5
Kopiere Register Hauptthread
S6
fork
S7a
Weiterarbeiten Hauptthread
S7b
Erzeugen/Start Zusatzthread
1
1
1
1

Figur 4


Weiterarbeiten Hauptthread
S7a
Erzeugen/Start Zusatzthread
S7b
Valid
Same
1
1
Ändern Register Hauptthread
S8
1
0
Ändern Register Zusatzthread
S9
stop
S10a
0
0
Beenden Zusatzthread
S10b
Beenden Hauptthread
S11