(12) DEMANDE INTERNATIONALE PUBLIÉE EN VERTU DU TRAITÉ DE COOPÉRATION EN MATIÈRE DE BREVETS (PCT)

(19) Organisation Mondiale de la Propriété Intellectuelle
Bureau international

(43) Date de la publication internationale
17 août 2017 (17.08.2017)

WIPO | PCT

(10) Numéro de publication internationale
WO 2017/137675 A1

(51) Classification internationale des brevets : *G06F 9/30* (2006.01)

(21) Numéro de la demande internationale : PCT/FR2017/050107

(22) Date de dépôt international : 19 janvier 2017 (19.01.2017)

(25) Langue de dépôt : français

(26) Langue de publication : français

(30) Données relatives à la priorité :
16 51063 10 février 2016 (10.02.2016) FR

(71) Déposant : **UPMEM** [FR/FR]; 2 Square Roger Génin, 38000 Grenoble (FR).

(72) Inventeurs : **DEVAUX, Fabrice**; C/O uPmem, 2 Square Roger Génin, 38000 Grenoble (FR). **FURODET, David**; C/O uPmem, 2 Square Roger Génin, 38000 Grenoble (FR).

(74) Mandataires : **DE JONG, Jean Jacques** et al.; Cabinet Omnipat, 24 Place des Martyrs de la Résistance, 13100 Aix En Provence (FR).

(81) États désignés *(sauf indication contraire, pour tout titre de protection nationale disponible)* : AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) États désignés *(sauf indication contraire, pour tout titre de protection régionale disponible)* : ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), eurasien (AM, AZ, BY, KG, KZ, RU, TJ, TM), européen (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

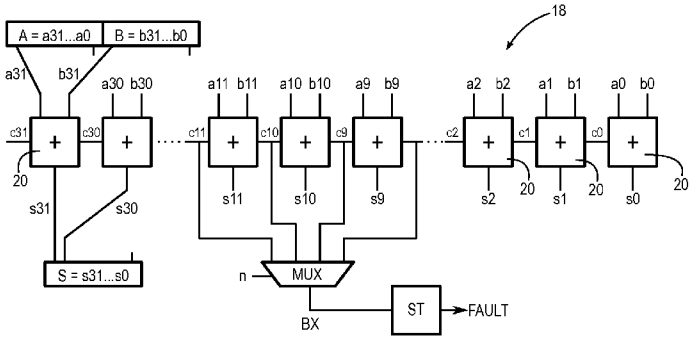
Publiée :

— *avec rapport de recherche internationale (Art. 21(3))*

(54) **Title** : COMBINED INSTRUCTION FOR ADDITION AND CHECKING OF TERMINALS

(54) **Titre** : INSTRUCTION COMBINÉE D'ADDITION ET DE VÉRIFICATION DE BORNES

Fig 2
ADDCK S, A, B, n



(57) **Abstract** : The invention relates to a processor core comprising, in the set of instructions thereof, a combined instruction for addition and checking of terminals (ADDCK) implicitly or explicitly defining an integer *n* as an instruction parameter; an adder having a width strictly greater than *n* bits; and a processing circuit (MUX, 42) designed to respond to the combined instruction by activating a crossing signal (BX) when an addition generates a carried number of rank n.

(57) **Abrégé** : L'invention est relative à un cœur de processeur comprenant, dans son jeu d'instructions, une instruction combinée d'addition et de vérification de bornes (ADDCK) définissant un entier *n* de manière implicite, ou de manière explicite comme un paramètre de l'instruction; un additionneur ayant une largeur strictement supérieure à *n* bits; et un circuit de traitement (MUX, 42) conçu pour répondre à l'instruction combinée par l'activation d'un signal de franchissement (BX) lorsqu'une addition génère une retenue de rang n.

WO 2017/137675 A1

INSTRUCTION COMBINÉE D'ADDITION ET DE VÉRIFICATION DE BORNES

Domaine

L'invention est relative aux transferts de données entre deux entités, notamment dans un cas général où la consommation de données par l'une des entités n'a pas la même granularité que la production de données par l'autre entité.

Arrière-plan

La figure 1 illustre un exemple de situation de transfert de données selon des granularités différentes entre une entité productrice 10 et une entité consommatrice 12. L'entité productrice 10 peut comprendre une mémoire système SMEM, par exemple de type dynamique, stockant une succession d'unités de données U à traiter par l'entité consommatrice 12. L'entité consommatrice peut comprendre un cœur de processeur CPU conçu pour lire les unités de données à traiter dans une mémoire de travail locale WMEM.

Une unité de données U comprend de façon générale plusieurs octets. Une unité de données est, par exemple, une « structure » définie d'après le langage utilisé pour programmer le processeur. Une telle structure est définissable de façon arbitraire par le programmeur pour contenir des variables de types et longueurs différents.

Le fait que les unités de données puissent être produites avec une granularité différente de celle utilisée pour leur consommation pose des difficultés pour réaliser une coordination efficace entre production et consommation.

Résumé

On prévoit de façon générale un cœur de processeur comprenant, dans son jeu d'instructions, une instruction combinée d'addition et de vérification de bornes définissant un entier *n* associé de manière implicite, ou de manière explicite comme un paramètre de l'instruction ; un additionneur ayant une largeur strictement supérieure à *n* bits ; et un circuit de traitement conçu pour répondre à l'instruction combinée par l'activation d'un signal de franchissement lorsque l'additionneur génère une retenue de rang *n*.

Le circuit de traitement peut être conçu en outre pour déclencher une faute d'exécution lorsque le signal de franchissement est activé, d'où il résulte que la procédure déclenchée pour traiter la faute peut être une procédure de gestion du débordement.

Selon une alternative, le circuit de traitement peut être conçu en outre pour provoquer un saut systématique, permettant ainsi de réaliser un saut en avant systématique d'une ou plusieurs instructions lorsque le signal de franchissement est inactif, et une continuation normale lorsque le signal de franchissement est activé, d'où il résulte que la ou les instructions suivant immédiatement l'instruction combinée peuvent être conçues pour gérer le franchissement.

Selon une autre alternative, le circuit de traitement peut être conçu en outre pour provoquer un appel de sous-programme lorsque le signal de franchissement est activé, et une continuation normale lorsque le signal de franchissement est inactif, d'où il résulte que le sous-programme peut être conçu pour gérer le franchissement.

Le cœur de processeur peut comprendre en outre dans son jeu d'instructions des instructions de lecture avec pré- et post-incrémentation d'adresse, incluant pour l'incrémentation d'adresse une addition combinée avec une vérification de bornes.

On prévoit également un procédé de transfert de données utilisant une instruction combinée d'addition et de vérification de bornes, comprenant les étapes suivantes mises en œuvre par le processeur : allouer une page mémoire, dont la taille est une puissance de 2 et dont l'adresse est un multiple de la taille, dans une mémoire de travail du processeur, pour contenir une série d'unités de données dans des emplacements consécutifs respectifs ; identifier le début d'un emplacement courant d'unité de données par un pointeur contenu dans un registre de travail du processeur ; utiliser l'emplacement courant ; exécuter une instruction combinée d'addition et de vérification de bornes pour incrémenter le pointeur, la borne de l'instruction combinée étant basée sur la taille de la page ; et exécuter une procédure de gestion de franchissement en réponse à l'activation du signal de franchissement.

La procédure de gestion de franchissement peut être programmée pour allouer une nouvelle page dans la mémoire de travail et mettre à jour le pointeur pour identifier un emplacement de départ dans la nouvelle page.

L'étape d'utilisation de l'emplacement courant dans la page mémoire peut comprendre la lecture de l'unité de données contenue dans l'emplacement courant pour la transférer dans des registres du processeur, le procédé comprenant en outre les étapes consistant à transférer dans la page mémoire des unités de données fournies par une entité productrice, les unités de données ayant une taille arbitraire pouvant entraîner que la page ne contienne pas un nombre entier d'unités de données, d'où il résulte que

l'activation du signal de franchissement peut se produire alors que des données ont été lues au-delà de la page ; réserver une zone mémoire à la suite de la page, de taille suffisante pour contenir une unité de données complète ; et transférer un nombre entier d'unités de données suffisant à remplir la page et déborder dans la zone mémoire réservée, d'où il résulte que les données lues au-delà de la page correspondent à des données effectives d'une unité de données.

Description sommaire des dessins

Des modes de réalisation seront exposés dans la description suivante, faite à titre non limitatif en relation avec les figures jointes parmi lesquelles :

- la figure 1, précédemment décrite, illustre un exemple de situation de transfert de données selon des granularités différentes entre une entité productrice et une entité consommatrice ;

- la figure 2 représente schématiquement un additionneur inclus dans une unité de traitement d'un cœur de processeur, complété selon un premier mode de réalisation pour offrir une instruction combinée d'addition et de vérification de bornes ; et

- les figures 3A et 3B illustrent une transition entre deux phases de lecture d'une mémoire provoquée par une instruction combinée d'addition et de vérification.

Description détaillée

Dans une situation telle qu'illustrée à la figure 1, le cœur de processeur CPU peut être conçu pour lire les unités de données U octet par octet dans la mémoire WMEM sans perte d'efficacité, alors qu'il ne serait pas envisageable de lire les unités de données octet par octet dans la mémoire système SMEM. Pour rentabiliser les accès à la mémoire système, souvent de type dynamique, les transferts sont généralement effectués par rafales de 2^n octets contigus, par exemple 256, 512 ou 1024 octets.

Dans l'exemple de la figure 1, on suppose qu'une rafale de transfert contient un peu plus de trois unités de données U. Ainsi, une première rafale permet de transférer dans la mémoire WMEM les unités U0 à U2 et le début de l'unité U3

Le processeur CPU lit les unités de données dans la mémoire WMEM octet par octet, par exemple. En atteignant l'unité U3, qui est incomplète, il est souhaitable que le processeur détecte une condition de franchissement et initie une nouvelle rafale de

transfert, contenant l'unité U3 complète et les unités suivantes (U4, U5, et le début de l'unité U6).

La nouvelle rafale est placée dans la mémoire WMEM dans la continuité de la rafale précédente, comme cela est représenté. Si la fin de la mémoire WMEM est atteinte, la mémoire est utilisée de façon circulaire, l'écriture des données continuant au début de la mémoire WMEM.

On appelle ci-après une « page » une zone de 2^n octets allouée dans la mémoire WMEM pour contenir une série d'unités de données en cours de traitement par le processeur. Une page peut être définie pour contenir une ou plusieurs rafales de transfert.

On reproduit ci-dessous un exemple d'opérations que l'on pourrait utiliser pour lire les unités de données et gérer le franchissement de page dans un cœur de processeur classique. On utilise à titre d'exemple un jeu d'instructions disponible sur les cœurs ARM.

```
; code 1
LDR Rt, [Rptr, #offset]
LDR Rt2, [Rptr, #offset2]
LDR Rt3, [Rptr, #offset3]
...
(Opérations sur les registres Rt, Rt2, Rt3...)
ADD Rptr, Rptr, Rinc
CMP Rmax, Rptr
BLMI [adresse routine de gestion de franchissement]
(Continuer lecture unités de données)
```

On désigne par Rptr un registre contenant le pointeur vers le début de l'unité de données U à lire dans la mémoire de travail WMEM. Ce registre a préalablement été chargé avec l'adresse correspondante.

Les octets contenus dans l'unité de données courante U sont alors lus par une succession d'instructions notées LDR (pour « LoaD Register ») sous la forme :

LDR Rt, [Rptr, #offset]

Où Rt est le registre destination, et #offset est une valeur immédiate qui est ajoutée aux bits de poids faible, appelés « offset », de l'adresse contenue dans le registre Rptr pour identifier l'octet à lire ou le mot de plusieurs octets à lire.

Une fois que l'unité de données courante a ainsi été transférée dans les registres du processeur, elle peut être soumise aux opérations que le programmeur lui a réservées, non explicitées ici.

Pour lire l'unité de données suivante, le registre Rptr est d'abord incrémenté de la taille de l'unité de données qui vient d'être lue, contenue par exemple dans un registre Rinc :

ADD Rptr, Rptr, Rinc

L'incrément Rinc est la taille, en octets, de l'unité de données U. Cet incrément peut être fixe ou être différent pour chaque unité de données.

Une telle mise à jour du pointeur implique que les unités de données sont lues séquentiellement. Les unités de données pourraient être lues dans un ordre arbitraire, auquel cas le pointeur serait mis à jour par d'autres opérations.

En tout cas, le nouveau pointeur contenu dans le registre Rptr peut encore être dans la limite de la page, cependant si près de la limite que la nouvelle unité de données à lire est tronquée et inexploitable (comme l'unité de données U3 à la figure 1). Pour gérer cette situation, on peut exécuter les instructions suivantes à chaque incrémentation du pointeur :

CMP Rmax, Rptr

Le contenu du registre Rptr est comparé au contenu d'un registre Rmax. En pratique cette instruction soustrait le contenu du registre Rptr au contenu du registre Rmax, et met à jour des variables d'état ou drapeaux du processeur pour refléter le résultat de la soustraction. Le registre Rmax peut contenir l'adresse de la fin de la page, diminuée de la taille de l'unité de données à lire. Ainsi, si la différence est positive ou nulle, l'unité de données suivante est entièrement contenue dans la page, et la lecture peut continuer par les opérations LDR précédemment décrites.

Si le résultat est négatif, ce qui est identifié par l'activation d'un drapeau noté MI, le contenu restant de la page n'est pas exploitable. On peut prévoir après l'instruction CMP une instruction d'appel conditionnel de sous-programme, sur la base du drapeau MI, par exemple l'instruction notée BLMI. Le sous-programme est alors conçu pour

gérer le franchissement, par exemple en transférant les unités de données suivantes dans une nouvelle page de la mémoire de travail.

La vérification de franchissement de la page implique ici l'exécution d'une paire d'instructions, comparaison (CMP) et appel conditionnel (BLMI), à chaque fois qu'on lit une nouvelle unité de données dans la mémoire de travail, ce qui peut diminuer l'efficacité du code, notamment lorsque les unités de données sont de petite taille.

Le brevet US 6542990 décrit une instruction de vérification de bornes BNDCHK qui combine une comparaison et la génération d'une faute d'exécution selon le résultat de la comparaison. La faute est traitée par le système d'exploitation pour gérer la cause du débordement. Une telle instruction pourrait être exécutée à la place de la paire d'instructions CMP et BLMI après chaque addition ADD. L'instruction BNDCHK est cependant conçue pour effectuer des comparaisons dans différentes configurations, résultant en une unité de traitement complexifiée.

Pour augmenter davantage l'efficacité du code, on prévoit ci-après d'ajouter une nouvelle instruction générique au jeu d'instructions du processeur, appelée instruction combinée d'addition et de vérification de bornes, notée ADDCK. Cette instruction, une évolution de la simple addition ADD, peut être conçue pour effectuer une addition en vérifiant simultanément que le résultat de l'addition ne franchit pas une borne strictement inférieure à la capacité de l'additionneur. (Par « simultanément », on entend que la vérification est effectuée dans le même cycle d'instruction que l'addition, c'est-à-dire que la vérification n'est pas effectuée par une instruction distincte.) La borne peut être implicite, c'est-à-dire associée à l'instruction, ou explicitement définie par un paramètre de l'instruction.

Plus précisément, la borne peut être exprimée sous la forme 2^n, où n est un entier. Si la capacité de l'additionneur est exprimée par 2^p, où p est la largeur de l'additionneur (typiquement 32 ou 64), on a $n < p$. Le franchissement d'une borne 2^n peut être détecté par l'activation du bit de retenue (« carry ») de rang n de l'additionneur, ci-après noté c_n. En utilisant un tel bit de retenue comme signal de détection, on définit en fait une multitude de bornes incluant 2^n et tous ses multiples jusqu'à 2^p. La « borne » devient alors la taille d'un segment, et l'instruction ADDCK détecte chaque transition d'un segment au segment contigu suivant de même taille dans un espace d'évolution. Cela est particulièrement utile dans des applications envisagées, où on souhaite notamment surveiller l'évolution d'un pointeur dans un espace d'adresses à cheval sur plusieurs pages contiguës de taille 2^n.

Si la vérification est satisfaite, c'est à dire si la borne n'est pas franchie par le résultat de l'addition, l'instruction ADDCK se comporte comme une simple addition ADD, et le programme se poursuit normalement.

Si la vérification échoue, l'instruction active un signal de franchissement qui, comme on l'a précédemment indiqué, peut être prélevé sur un bit de retenue dont le rang n est défini implicitement ou par un paramètre de l'instruction. Le signal de franchissement peut être exploité de plusieurs manières. Il peut activer une variable d'état ou drapeau du processeur, lequel drapeau est désactivé lorsque le franchissement a été géré.

On peut noter qu'un additionneur classique peut être conçu pour activer un drapeau de débordement (« overflow ») lorsque le résultat de l'addition dépasse la capacité de l'additionneur. Ce drapeau correspond généralement à la retenue de dernier rang p de l'additionneur. L'additionneur étant généralement conçu pour traiter les nombres les plus grands gérés par le processeur, sa capacité 2^p est bien trop grande pour être utilisable comme borne dans les applications envisagées pour l'instruction ADDCK. L'instruction ADDCK utilise une borne strictement inférieure à la capacité de l'additionneur, et généralement significativement plus petite. Par exemple, dans le cadre d'un processeur avec un additionneur de 32 bits, donc ayant une capacité de 2^{32}, les bornes associées à l'instruction ADDCK peuvent, par exemple, aller de 2^8 à 2^{12}.

L'activation d'un drapeau de franchissement par l'instruction ADDCK peut provoquer une faute d'exécution qui cède immédiatement la main au système d'exploitation pour gérer le franchissement. Le système d'exploitation cède ensuite la main au programme qui continue au point où il en était.

Le drapeau de franchissement peut générer une interruption du processeur, interruption qui peut être gérée de manière similaire à une faute d'exécution.

Ces deux alternatives (faute et interruption) sont relativement coûteuses en cycles d'instruction, car elles font appel au système d'exploitation. On préfèrera des alternatives qui appellent directement une procédure de gestion de franchissement.

Ainsi, au lieu d'activer un drapeau, le signal de franchissement peut provoquer l'appel d'un sous-programme à une adresse implicite ou fournie comme paramètre de l'instruction ADDCK. Le sous-programme est alors prévu par le programmeur pour gérer le franchissement.

Selon une autre alternative, l'absence d'activation du signal de franchissement peut provoquer un saut inconditionnel, notamment un saut en avant d'un nombre d'instructions implicite ou défini par un paramètre de l'instruction ADDCK. L'activation du signal de franchissement n'a alors aucun effet, de sorte que l'exécution continue avec les instructions suivant immédiatement l'instruction ADDCK, lesquelles instructions sont alors prévues pour gérer le franchissement. Le saut peut être d'une seule instruction, auquel cas on prévoit un appel de sous-programme immédiatement après l'instruction ADDCK.

Une instruction ADDCK est générique et peut avoir de nombreux usages. Elle a été conçue à l'origine pour augmenter l'efficacité du code dans des situations du type de la figure 1. Dans l'exemple « code 1 » ci-dessus, l'instruction d'addition simple :

ADD Rptr, Rptr, Rinc

est alors remplacée par une instruction combinée d'addition et de vérification de bornes :

ADDCK Rptr, Rptr, Rinc, n

Si l'instruction ADDCK est conçue pour provoquer un saut, l'adresse du saut peut être implicite (par exemple l'adresse deux instructions plus loin en absence de franchissement) auquel cas l'instruction n'a pas besoin de paramètre supplémentaire. Si l'adresse du saut est explicite, elle pourra être fournie en cinquième paramètre ou comme « méta-paramètre », c'est-à-dire que l'instruction est alors conçue pour utiliser un registre dédié contenant l'adresse du saut.

Les instructions de comparaison CMP et d'appel conditionnel de sous-programme BLMI de l'exemple « code 1 » sont alors omises, et cela pour chaque itération de lecture d'une unité de données. On obtient à titre d'exemple les opérations suivantes :

```
; code 2
LDR Rt, [Rptr, #offset]
LDR Rt2, [Rptr, #offset2]
LDR Rt3, [Rptr, #offset3]
...
(Opérations sur les registres Rt, Rt2, Rt3...)
ADDCK Rptr, Rptr, Rinc, n
(Continuer lecture unités de données)
```

La figure 2 illustre schématiquement un premier mode de réalisation de circuit de vérification construit autour de l'additionneur 18 présent dans l'unité arithmétique d'un processeur simple. L'instruction à exécuter est notée

ADDCK S, A, B, n

où S désigne le registre recevant le résultat de l'addition, A et B désignent les registres contenant les opérandes, et *n* est le logarithme en base 2 de la taille de la page courante. En d'autres termes, la taille de la page est égale à 2^n octets. Les bits contenus dans les registres S, A et B sont désignés par s_i, a_j, b_k, où *i*, *j* et *k* sont des entiers désignant les poids des bits.

A titre d'exemple, l'additionneur 18 est un additionneur de 32 bits comprenant 32 cellules d'addition élémentaires 20. Pour simplifier l'exposé, on suppose que l'additionneur est de type à propagation de retenue (« ripple-carry »). Chaque cellule 20, sauf la première, est un additionneur complet ayant deux entrées pour les opérandes a_i, b_i, une entrée de retenue c_{i-1}, une sortie de résultat s_i, et une sortie de retenue c_i reliée à l'entrée de retenue de la cellule suivante. La première cellule diffère en ce qu'elle n'a pas d'entrée de retenue.

Comme on l'a précédemment indiqué, le signal de franchissement BX peut être prélevé sur le bit de retenue c_n.

Si on choisit une taille de page fixe pour le système, par exemple de $2^{10} = 1024$ octets, le paramètre *n* de l'instruction ADDCK peut être omis et le signal de franchissement BX est directement prélevé sur le bit de retenue c_{10}.

On notera que le rang *n* du bit de retenue servant à fournir ainsi le signal de franchissement est généralement petit devant la largeur de l'additionneur, et en tout cas strictement inférieur à la largeur de l'additionneur.

La figure 2 illustre plus en détail un exemple permettant de mettre en œuvre un paramètre *n* variable de 8 à 11, et donc de choisir des pages de 256, 512, 1024 ou 2048 octets. Le bit de retenue c_8 à c_{11} à utiliser comme signal de franchissement BX est sélectionné par un multiplexeur MUX commandé par le paramètre *n*. Comme on l'a déjà mentionné, le signal BX peut activer une variable d'état ST du processeur et, par

exemple, déclencher une faute d'exécution servant à lancer une procédure de gestion du franchissement.

Avec cette structure, lorsque le pointeur identifie une unité de données incomplète à la fin de la page, comme l'unité U3 à la figure 1, le franchissement n'est pas encore signalé, puisque le pointeur est dans les limites de la page. Le franchissement sera signalé à l'incrémentation suivante, alors que l'unité de données incomplète (U3) aura été lue par le processeur. Dans ce cas, la procédure de gestion de franchissement pourrait être conçue pour annuler les effets de la lecture de la dernière unité de données. Pour éviter cette complication, on peut procéder de la manière exposée ci-après.

Les figures 3A et 3B illustrent une transition entre la lecture d'une page et la préparation de la page suivante à la suite de l'exécution d'une instruction ADDCK.

A la figure 3A une première page PG1 est allouée dans la mémoire WMEM, par exemple en partant du début de la mémoire. Les unités de données sont transférées dans la mémoire par rafales ou unités de transfert TU dont la taille est choisie pour être un sous-multiple de la taille d'une page, pour contenir complètement au moins une unité de données, et pour offrir une efficacité de transfert suffisante à partir de l'entité productrice (par exemple une mémoire dynamique). A titre d'exemple, la taille de la page peut être de 1024 octets et la taille des unités de transfert peut être de 256 octets.

A la figure 3A, on a écrit dans la mémoire WMEM un nombre suffisant d'unités de transfert TU (par exemple 4) pour former la page PG1, et une unité de transfert en plus. L'ensemble des unités de transfert écrites (cinq) contient les unités de données U0 à U10. L'unité U8 déborde de la fin de la page PG1, mais elle est entièrement contenue dans la mémoire WMEM.

L'instruction ADDCK est utilisée pour incrémenter un pointeur ptr à chaque lecture d'une unité de données U. Lorsque le pointeur ptr arrive à l'emplacement de l'unité U8, l'instruction ADDCK ne signale pas encore de franchissement, puisque le pointeur est dans les limites de la page. La lecture de l'unité U8 démarre dans la page et se termine hors de la page. L'instruction ADDCK est ensuite exécutée pour incrémenter le pointeur ptr, qui identifie alors l'emplacement contenant l'unité U9. Le pointeur est alors hors de la page et le franchissement est signalé.

A la figure 3B, la procédure de gestion de franchissement alloue une nouvelle page PG2 dans la mémoire WMEM à la suite de la page PG1, et lance l'écriture des données suivantes dans la nouvelle page. Le transfert démarre à partir du point où l'unité de

données U8 avait été tronquée dans la première page. Cependant, comme l'unité U8 a été correctement prise en compte par le processeur, le pointeur ptr est mis à jour pour démarrer sur l'unité de données U9.

Jusqu'ici on a supposé que les unités de données U pouvaient avoir une taille variable, rendant difficile la prédiction du débordement de la page et impliquant une gestion d'unités de données pouvant être incomplètes (U8). Si toutes les unités de données ont la même taille en mémoire de 2^m octets (par exemple 64), alors une page de 2^n octets (par exemple 1024) contient exactement 2^{n-m} unités de données complètes (par exemple 16).

Dans ce cas, lorsque le pointeur déborde de la page, il identifie l'emplacement suivant immédiatement la page. Il en résulte que la dernière unité de données lue était contenue complètement dans la page et qu'une procédure du type des figures 3A et 3B est inutile.

Dans certains processeurs, il existe des instructions qui combinent une addition à une autre opération qui n'a pas besoin de l'additionneur. Par exemple, le jeu d'instructions ARM comprend des instructions de lecture (LDR) et écriture (STR) avec pré- ou post-incrémentation du registre stockant l'adresse de lecture ou écriture. Avec une pré-incrémentation, le registre d'adresse est d'abord incrémenté et sert ensuite à accéder à la mémoire. Avec une post-incrémentation, le registre d'adresse est d'abord utilisé pour accéder à la mémoire et incrémenté ensuite. Ces instructions peuvent être doublées de versions offrant la vérification de bornes en association avec l'addition utilisée pour incrémenter le registre d'adresse.

On a décrit jusqu'ici des exemples de transfert de données où le processeur était consommateur d'unités de données. Le processeur pourrait également être producteur d'unités de données à transférer dans la mémoire système, auquel cas, dans les exemples de code fournis, les instructions de lecture dans la mémoire de travail (LDR) sont remplacées par des instructions d'écriture dans la mémoire de travail (STR). L'instruction ADDCK peut être utilisée de manière similaire. La procédure de gestion de franchissement peut alors être prévue pour vider le contenu de la page courante et allouer une page vierge pour continuer l'écriture.

De nombreuses variantes et modifications des modes de réalisation décrits ici apparaîtront à l'homme du métier. Par exemple, bien que l'on ait décrit un pointeur qui est incrémenté pour parcourir des adresses croissantes, on peut également envisager un pointeur qui est décrémenté pour parcourir des adresses décroissantes. L'opération de

décrémentation n'est autre qu'une addition d'un incrément négatif à la valeur du pointeur, qui utilise une structure d'additionneur conçue pour travailler sur des nombres binaires signés. Dans le cas d'une soustraction, le signal de franchissement BX dans la structure de la figure 2 est fourni par le complément du bit de retenue.

5

Revendications

1. Cœur de processeur comprenant :

   - dans son jeu d'instructions, une instruction combinée d'addition et de vérification de bornes (ADDCK) définissant un entier *n* de manière implicite, ou de manière explicite comme un paramètre de l'instruction ;
   - un additionneur ayant une largeur strictement supérieure à *n* bits ; et
   - un circuit de traitement (MUX, 42) conçu pour répondre à l'instruction combinée par l'activation d'un signal de franchissement (BX) lorsque l'additionneur génère une retenue de rang *n*.

2. Cœur de processeur selon la revendication 1, dans lequel le circuit de traitement est conçu pour déclencher une faute d'exécution lorsque le signal de franchissement est activé, d'où il résulte que la procédure déclenchée pour traiter la faute peut être une procédure de gestion du débordement.

3. Cœur de processeur selon la revendication 1, dans lequel le circuit de traitement est conçu pour provoquer un saut en avant systématique d'une ou plusieurs instructions lorsque le signal de franchissement est inactif, et une continuation normale lorsque le signal de franchissement est activé, d'où il résulte que la ou les instructions suivant immédiatement l'instruction combinée peuvent être conçues pour gérer le franchissement.

4. Cœur de processeur selon la revendication 1, dans lequel le circuit de traitement est conçu pour provoquer un appel de sous-programme lorsque le signal de franchissement est activé, et une continuation normale lorsque le signal de franchissement est inactif, d'où il résulte que le sous-programme peut être conçu pour gérer le franchissement.

5. Cœur de processeur selon la revendication 1, comprenant en outre dans son jeu d'instructions des instructions de lecture avec pré- et post-incrémentation d'adresse, incluant pour l'incrémentation d'adresse une addition combinée avec une vérification de bornes.

6. Procédé de transfert de données utilisant une instruction combinée d'addition et de vérification de bornes (ADDCK) selon la revendication 1, comprenant les étapes suivantes mises en œuvre par le processeur :

- allouer une page mémoire, dont la taille est une puissance de 2 et dont l'adresse est un multiple de la taille, dans une mémoire de travail du processeur, pour contenir une série d'unités de données (U) dans des emplacements consécutifs respectifs ;
- identifier le début d'un emplacement courant d'unité de données par un pointeur contenu dans un registre de travail (Rptr) du processeur ;
- utiliser l'emplacement courant ;
- exécuter une instruction combinée d'addition et de vérification de bornes (ADDCK) pour incrémenter le pointeur, la borne de l'instruction combinée étant basée sur la taille de la page ; et
- exécuter une procédure de gestion de franchissement en réponse à l'activation du signal de franchissement.

7. Procédé selon la revendication 6, dans lequel la procédure de gestion de franchissement est programmée pour allouer une nouvelle page dans la mémoire de travail et mettre à jour le pointeur pour identifier un emplacement de départ dans la nouvelle page.

8. Procédé selon la revendication 6, dans lequel l'étape d'utilisation de l'emplacement courant dans la page mémoire comprend la lecture de l'unité de données contenue dans l'emplacement courant pour la transférer dans des registres du processeur, le procédé comprenant en outre les étapes suivantes :

- transférer dans la page mémoire des unités de données fournies par une entité productrice, les unités de données ayant une taille arbitraire pouvant entraîner que la page ne contienne pas un nombre entier d'unités de données, d'où il résulte que l'activation du signal de franchissement peut se produire alors que des données ont été lues au-delà de la page ;
- réserver une zone mémoire à la suite de la page, de taille suffisante pour contenir une unité de données complète ; et
- transférer un nombre entier d'unités de données suffisant à remplir la page et déborder dans la zone mémoire réservée, d'où il résulte que les données lues au-delà de la page correspondent à des données effectives d'une unité de données.

Fig 1

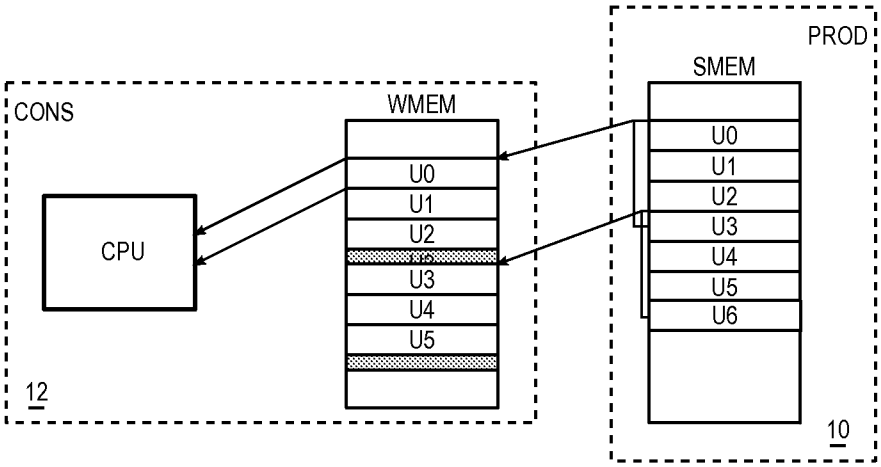
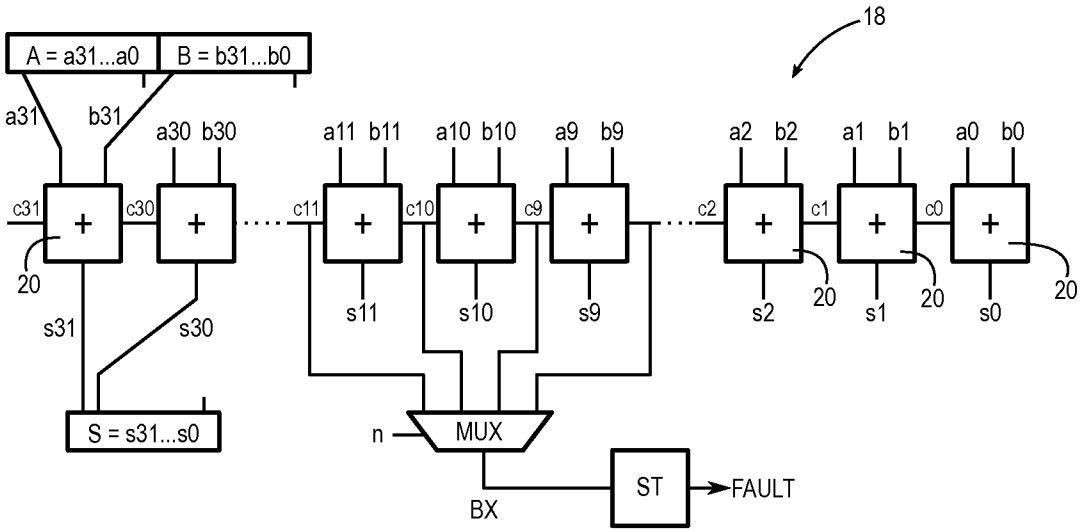


Fig 2

ADDCK S, A, B, n



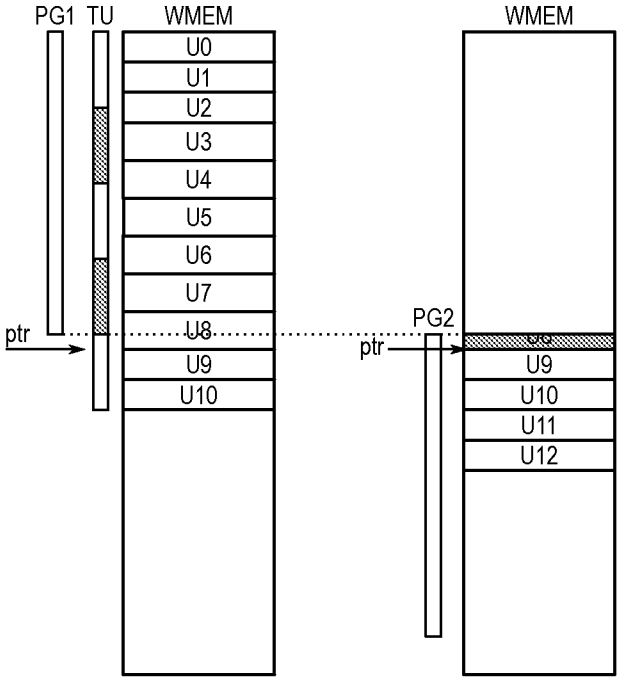

PG1
TU
WMEM
U0
U1
U2
U3
U4
U5
U6
U7
U8
U9
U10
ptr
WMEM
PG2
ptr
U9
U10
U11
U12

Fig 3A Fig 3B

INTERNATIONAL SEARCH REPORT

International application No
PCT/FR2017/050107

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/30
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	Intel: "Intel 64 and IA-32 Architectures Software Developer's Manual snippets", , 1 September 2006 (2006-09-01), XP055292424, Retrieved from the Internet: URL:http://www.intel.com/ [retrieved on 2016-08-01]	1-5
Y	the whole document	6-8

A	US 6 542 990 B1 (TREMBLAY MARC [US] ET AL) 1 April 2003 (2003-04-01) cited in the application the whole document	1-8

Y	EP 2 096 551 A1 (FUJITSU LTD [JP]) 2 September 2009 (2009-09-02) the whole document	6-8

	-/--	

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
14 March 2017	23/03/2017

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Ghidini, Mario

2

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	David Kanter: "Intel's Haswell CPU Microarchitecture", , 13 November 2012 (2012-11-13), XP055354045, Retrieved from the Internet: URL:http://www.realworldtech.com/haswell-cpu/4/ [retrieved on 2017-03-13] the whole document -----	1-8

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/FR2017/050107

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 6542990 B1	01-04-2003	US 6408383 B1	18-06-2002
		US 6542990 B1	01-04-2003
EP 2096551 A1	02-09-2009	CN 101520760 A	02-09-2009
		EP 2096551 A1	02-09-2009
		JP 5226341 B2	03-07-2013
		JP 2009205366 A	10-09-2009
		KR 20090092722 A	01-09-2009
		US 2009216919 A1	27-08-2009

RAPPORT DE RECHERCHE INTERNATIONALE

Demande internationale n°
PCT/FR2017/050107

A. CLASSEMENT DE L'OBJET DE LA DEMANDE
INV. G06F9/30
ADD.

Selon la classification internationale des brevets (CIB) ou à la fois selon la classification nationale et la CIB

B. DOMAINES SUR LESQUELS LA RECHERCHE A PORTE

Documentation minimale consultée (système de classification suivi des symboles de classement)
G06F

Documentation consultée autre que la documentation minimale dans la mesure où ces documents relèvent des domaines sur lesquels a porté la recherche

Base de données électronique consultée au cours de la recherche internationale (nom de la base de données, et si cela est réalisable, termes de recherche utilisés)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERES COMME PERTINENTS

Catégorie*	Identification des documents cités, avec, le cas échéant, l'indication des passages pertinents	no. des revendications visées
X	Intel: "Intel 64 and IA-32 Architectures Software Developer's Manual snippets", , 1 septembre 2006 (2006-09-01), XP055292424, Extrait de l'Internet: URL:http://www.intel.com/ [extrait le 2016-08-01]	1-5
Y	le document en entier	6-8

A	US 6 542 990 B1 (TREMBLAY MARC [US] ET AL) 1 avril 2003 (2003-04-01) cité dans la demande le document en entier	1-8

Y	EP 2 096 551 A1 (FUJITSU LTD [JP]) 2 septembre 2009 (2009-09-02) le document en entier	6-8

	-/--	

[X] Voir la suite du cadre C pour la fin de la liste des documents

[X] Les documents de familles de brevets sont indiqués en annexe

* Catégories spéciales de documents cités:

"A" document définissant l'état général de la technique, non considéré comme particulièrement pertinent

"E" document antérieur, mais publié à la date de dépôt international ou après cette date

"L" document pouvant jeter un doute sur une revendication de priorité ou cité pour déterminer la date de publication d'une autre citation ou pour une raison spéciale (telle qu'indiquée)

"O" document se référant à une divulgation orale, à un usage, à une exposition ou tous autres moyens

"P" document publié avant la date de dépôt international, mais postérieurement à la date de priorité revendiquée

"T" document ultérieur publié après la date de dépôt international ou la date de priorité et n'appartenant pas à l'état de la technique pertinent, mais cité pour comprendre le principe ou la théorie constituant la base de l'invention

"X" document particulièrement pertinent; l'invention revendiquée ne peut être considérée comme nouvelle ou comme impliquant une activité inventive par rapport au document considéré isolément

"Y" document particulièrement pertinent; l'invention revendiquée ne peut être considérée comme impliquant une activité inventive lorsque le document est associé à un ou plusieurs autres documents de même nature, cette combinaison étant évidente pour une personne du métier

"&" document qui fait partie de la même famille de brevets

Date à laquelle la recherche internationale a été effectivement achevée	Date d'expédition du présent rapport de recherche internationale
14 mars 2017	23/03/2017

Nom et adresse postale de l'administration chargée de la recherche internationale	Fonctionnaire autorisé
Office Européen des Brevets, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Ghidini, Mario

2

Formulaire PCT/ISA/210 (deuxième feuille) (avril 2005)

C(suite). DOCUMENTS CONSIDERES COMME PERTINENTS

Catégorie*	Identification des documents cités, avec, le cas échéant, l'indication des passages pertinents	no. des revendications visées
A	David Kanter: "Intel's Haswell CPU Microarchitecture", , 13 novembre 2012 (2012-11-13), XP055354045, Extrait de l'Internet: URL:http://www.realworldtech.com/haswell-cpu/4/ [extrait le 2017-03-13] le document en entier -----	1-8

RAPPORT DE RECHERCHE INTERNATIONALE

Renseignements relatifs aux membres de familles de brevets

Demande internationale n°
PCT/FR2017/050107

Document brevet cité au rapport de recherche	Date de publication	Membre(s) de la famille de brevet(s)	Date de publication
US 6542990 B1	01-04-2003	US 6408383 B1	18-06-2002
		US 6542990 B1	01-04-2003
EP 2096551 A1	02-09-2009	CN 101520760 A	02-09-2009
		EP 2096551 A1	02-09-2009
		JP 5226341 B2	03-07-2013
		JP 2009205366 A	10-09-2009
		KR 20090092722 A	01-09-2009
		US 2009216919 A1	27-08-2009