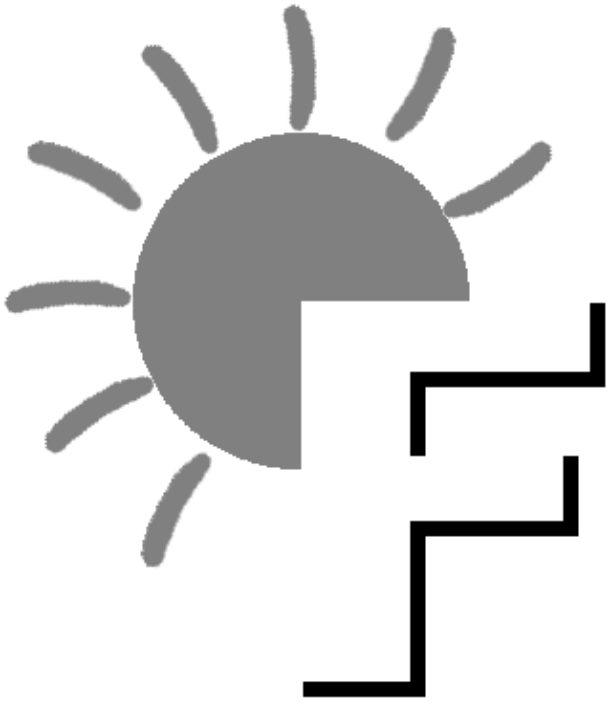
La GPL appliquée à l'industrie microélectronique : "propriété intellectuelle ouverte", le projet **Freedom CPU**



Brochure de présentation du projet F−CPU
(C) F−CPU Team, 2000

Sommaire :

A propos de ce document	3
Origine du projet :	3
Introduction au F−CPU actuel :	4
Organisation du groupe :	4
Architecture modulaire du f−cpu :	4
Caractéristiques principales du F−CPU actuel :	5
Introduction au FC0 :	7
Superpipeline :	7
Scoreboard/Xbar :	8
Exceptions :	8
3r1w/2r2w :	9
SRB :	10
Programmation :	11
Fabriquer un F−CPU :	11
Conclusion :	12

A propos de ce document

Ce papier est une brochure destinée à présenter le projet F−CPU, sa philosophie et les travaux réalisés dans ce cadre. Il résume certaines parties du manuel et le complémente par des discussions de nature plus politique. C'est la version originale qui sera traduite ensuite dans d'autres langues. Il est aussi disponible aux formats HTML, PDF et PostScript sur le site web principal du projet : http://www.f−cpu.org. Il a été écrit par Yann Guidon et Djamel Sebaihi et est distribué selon les termes de la GFDL (GNU Free Documentation Licence).

Dernière mise à jour : YG, 17/8/2000.

Origine du projet :

Le projet F−CPU est le point de convergence entre différents mouvements anciens et nouveaux, dont certains sont rendus possibles par le succès de l'idée du Logiciel Libre. Deux ans après la naissance de l'idée du F−CPU, le projet grandit et évolue malgré toutes les difficultés rencontrées. Certaines motivations de base sont par exemple :

- Le succès de Linux et l'adoption de la GPL comme modèle économique et remplacement efficace des shareware
- Sentiments anti−Intel/Microsoft, rejet des dérapages du système industriel vers l'accaparation de la propriété intellectuelle (prise des consommateurs en otage)
- Les nouvelles technologies (FPGA, ASIC) sont plus efficaces, moins chères et plus répandues.

Pour nous debarrasser du couple Wintel, nous devons logiquement créer un nouveau microprocesseur grand public et nous pouvons nous inspirer de l'exemple de GNU/Linux comme modèle de développement communautaire et de distribution. Cela est facile a priori, comme le montrent les nombreuses tentatives dans ce domaine (comme OpenCPU, LEON, E−RISC...) mais pour que le projet soit un succès, il a besoin d'un ingrédient très important, qui doit être bien dosé : UNE ARCHITECTURE.

La premiere architecture était "Memory to Memory" (M2M) : les initiateurs du projet F−CPU (introuvables aujourd'hui) considéraient que le problème le plus important était celui des changements de tâches, donc de la sauvegarde du contexte d'une application. Depuis, le problème a été résolu par la technique du "smooth register backup" (examiné plus loin) et ce n'est plus une contrainte architecturale importante.

Ensuite, une architecture dite "Transfer Triggered Architecture" (TTA) a été proposée. Après de nombreuses dicussions elle a posé des problèmes de flexibilité : elle impose un "jeu d'instructions jetable" qui oblige à recompiler tous les programmes lorsque le processeur évolue. Un coeur de CPU TTA pourrait devenir un jour une bonne alternative aux coeurs OOO classiques dans le cas où les instructions RISC sont traduites à le volée comme dans le PII ou les derniers MIPS mais le problème ne se pose pas actuellement.

Aujourd'hui le F−CPU utilise une architecture inspirée du modèle RISC classique. 20 ans après la création de ce concept, nous pouvons trier les caractéristiques qui ont "bien" ou "mal" vieilli, et l'ambition du projet est de créer une architecture meilleure que celle de l'Alpha de DEC. Le coeur actuel a beaucoup de place pour les améliorations futures et il est conçu pour être efficace et simple.

Introduction au F−CPU actuel :

Le F−CPU est un projet ouvert, sans véritable "leader", dont l'objectif philosophique est de développer une alternative crédible et ouverte aux architectures propriétaires comme celles d'Intel, Motorola, IBM... Ce projet n'a pas d'objectif imposé comme dans un projet industriel : il n y a donc pas de "deadline", pas de budget, pas d'obligation de résultat, et une liberté totale d'entrependre. C'est une sorte de laboratoire virtuel qui permet de tester des idées à plus grande échelle que dans un coin de son garage.

Le projet F−CPU est ce que les gens en font : ils peuvent participer à sa définition technique ou se le réapproprier selon leur goût et leur besoin sans se préoccuper d'objectifs corporatifs, à condition de respecter les termes de la licence F−CPU (inspirée de la GPL et actuellement en cours d'écriture à http://www.opencollector.org/hardlicense/fcpu.html). Comme avec la GPL, l'utilisateur a le droit d'utiliser, de comprendre, de modifier, d'améliorer et de redistribuer le F−CPU à la seule condition de conserver intacts les copyrights et la licence de distribution de la Propriété Intellectuelle (IP en anglais).

L'activite réelle du groupe consiste en des discussions sur la mailing list, axées sur les aspects techniques, philosophiques et organisationnels du projet. La "production" est donc principalement intellectuelle et peu de code a été écrit pour l'instant; la somme des conclusions des discussions est réunie dans le manuel ("F−CPU MANUAL", version 0.2 actuellement) qui décrit le projet et les détails architecturaux du processeur.

Il est probable que plusieurs autres architectures différentes verront le jour dans le cadre du projet F−CPU, le coeur du processeur actuel (FC0) est donc principalement une étude de cas dans le but de faire émerger le projet. Le FC0 sera sûrement remplacé plus tard par des coeurs plus performants et la mailing list discute déjà d'idées pour des alternatives.

Organisation du groupe :

Le projet F−CPU gravite autour d'une mailing list anglophone ouverte à toutes et à tous: f−cpu@egroups.com. C'est là que les détails architecturaux sont discutés et disséqués durant de long threads mouvementés (qui ressemblent un peu a ce qu'on peut lire sur USENET dans comp.arch).

Des mailings listes locales permettent aux allemand, aux français et aux espagnols de discuter des problèmes d'organisation dans leur pays et dans leur langue : ce sont les forums pour des structures légales à but non lucratif (association loi 1901 en France, Verein en Allemagne) qui font la promotion du projet dans leur pays et permettent de mieux repartir les tâches entre les membres.

Architecture modulaire du f−cpu :

Le F−CPU est une architecture destinée à changer constamment, elle est donc modulaire et chaque niveau doit pouvoir être interchangeable. Ceci est d'abord dû à des pressions internes des équipes de développement qui ne sont jamais d'accord à 100% entre elles, à des problèmes d'évolutivité (l'augmentation de puissance nécessite des remises en question constantes) et à des contraintes de protection de la Propriété Intellectuelle. Il est probable que des sociétés abusent des systèmes de brevet et de copyright, menaçant le projet et l'idée même de liberté dans le monde de l'électronique. La parade est simple, à la fois passive et active :

- publier, divulguer et informer le public, par tous les médias possibles, afin de constituer des preuves d'antériorité incontestables permettant de se défendre contre les éventuelles poursuites judiciaires

abusives. On ne peut pas breveter des systèmes qui ont déjà été divulgués, et il est impossible de breveter toutes les innovations du projet.
- Utiliser les armes simples et légales des copyright et des licences. Une "licence Free IP", similaire à la GPL, est en cours d'élaboration en collaboration avec d'autres groupes similaires.
- En cas de problème grave, la modularité permet d'adopter la "stratégie du lézard" : abandonner l'implémentation d'un module et en créer un nouveau (meilleur, tant qu'à faire).
- Dans tous les cas, l'équipe de développement du projet F−CPU ne manipule que des données abstraites et immatérielles, le seul problème possible serait un problème de copyright. Le groupe lui−même ne travaille pas sur l'implémentation et est à l'abris des problèmes de brevets.

Il n'est donc pas question de breveter quoique ce soit dans le projet car en pratique, c'est une protection limitée dans le temps, très chère et facilement contournable par des améliorations ou des généralisations. De plus, il arrive souvent que des brevets soient acceptés bien que les prétentions soient impossibles, ou identiques à d'autres brevets. Enfin, le projet F−CPU n'a pas d'argent ni de temps à perdre dans des problèmes non techniques et qui freinent la créativité du projet.

Les "modules" du projet sont répartis en plusieurs étages :

- codes sources en langage de haut niveau (par exemple pour coder un traitement de textes ou un compilateur). Ils doivent tous être distribués sous GPL, ou une licence "opensource" si ce n'est qu'un portage. C'est à ce niveau que les grands efforts logiciels auront lieu (quantitativement). Ce niveau est relativement indépendant du projet.
- codes sources en langage machine : tous codés sous GPL (ce n'est pas compliqué puisqu'il faut tout réécrire). Deux versions de la syntaxe assembleur existent déjà, pour deux usages différents (programmation manuelle ou backend de GCC). Ce niveau est dépendant du projet car il est lié à l'ISA par la forme des instructions. Certains niveaux sont préservés comme l'allocation des opcodes qui peut changer par un simple fichier #include différent.
- prorgammes compilés (binaires) : dépend directement de l'ABI et de l'ISA.
- ABI (Application Binary Interface) : toutes les conventions pour appeler une fonction système ou traiter les interruptions. Peu de définitions existent actuellement mais les conventions seront similaires à celles de l'Alpha ou du PowerPC en essayant de simplifier les mécanismes au maximum.
- ISA (Instruction Set Architecture) : définit les instructions, leurs opcodes, leur fonction, leur comportement, leur codage en binaire et les contraintes d'utilisation (champs réservés etc). C'est le coeur du système et occupe une grande partie du manuel.
- Coeurs de CPU : seul le FC0 est avancé dans le développement.
- Interface matérielle (sockets, entrées/sorties, contrôleurs de mémoire intégrés...) : des standards "open" seront utilisés mais les prototypes auront probablement une interface spécifique pour faciliter le développement. Pas de décisions définitives de ce côté−là (ce qui laisse beaucoup de liberté).

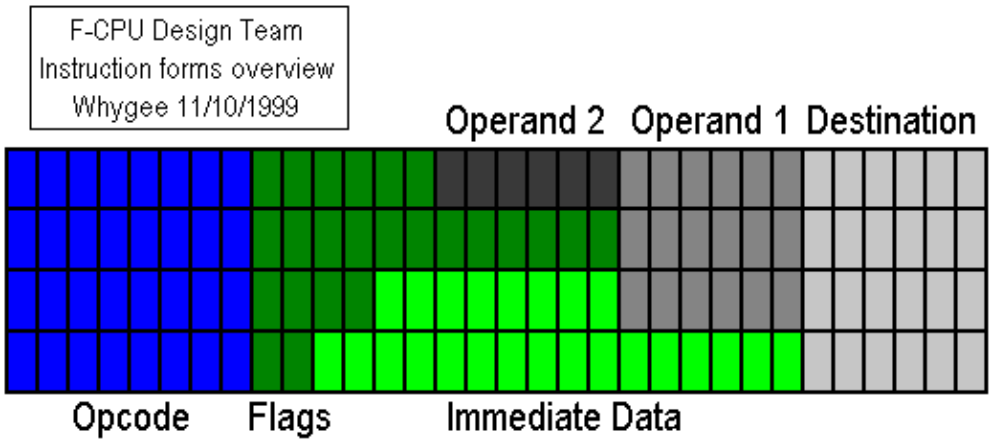
Caractéristiques principales du F−CPU actuel :

Tous les niveaux sont étudiés en même temps afin de simplifier la fabrication prochaine de prototypes mais des points de rupture ou d'indépendence conservent l'intéropérabilité entre des versions différentes de deux niveaux. Ainsi l'ABI n'est pas directement dépendente de la syntaxe du langage assembleur, et un coeur de CPU peut toujours en émuler un autre (voir le PII qui traduit les instructions en micro−opérations). Par contre les caractéristiques de base du F−CPU actuel sont importantes et influencent certaines couches :

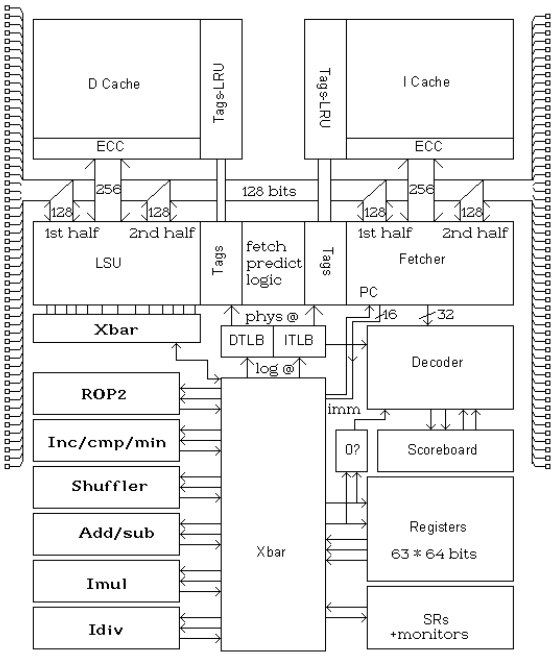
- Pas d'organisation VLIW car technologie trop récente et peu stable (on laisse Intel/HP se vautrer avec l'IA64)

- Instructions de type RISC sur 32 bits, 4 formats et opcode 8 bits, 120 opcodes sur 256 sont réservés.
- 64 registres "à tout faire" (int/FP/addresses) dont r0 qui est câblé à 0
- Les registres et les instructions sont naturellement SIMD et de largeur arbitrairement large (non limitée à 64 bits)
- La configuration s'effectue par des registres spéciaux séparés du banc de registres et accédés par deux instructions bloquantes seulement (PUT et GET)
- Pas de pointeur de pile dédié (comme pour l'Alpha)
- Pas de registre de code condition, mais des instructions conditionnelles (**jumpra** et **move**) qui testent si un registre est à zéro ou si le MSB ou le LSB est à zéro (ou à un).
- Pas de problème d'endian (bit endian dans l'instruction de load/store)
- Mémoire paginée (adresses virtuelles/physiques différentes, l'OS est responsable de gérer les TLB misses) avec un VMID par tâche
- Protection user/supervisor plus des bits spécifiques autorisant tâche par tâche l'utilisation des ressources matérielles (plus simple pour écrire des device drivers)

Les instructions utilisent uniquement 4 formats :



Introduction au FC0 :



Le FC0 est un premier coeur de CPU conçu à l'occasion du projet F−CPU mais il pourrait très bien exécuter des binaires de processeurs RISC similaires (MIPS, ALPHA, Power, ARM...) avec quelques adaptations (nombre de registres et décodeur d'instructions par exemple). Le FC0 se démarque cependant des autres CPU par certains paramètres dans l'étude de l'architecture : en analysant les autres coeurs RISC, on s'aperçoit qu'ils sont vieux de 10 à 20 ans et il est facile de voir les idées qui ont fonctionné (banc de registres monolithique et instructions de taille fixe par exemple) et celles qui ont eu des effets de bord plus tard (branchements retardés, pas assez de registres lors de l'explosion des coeurs superscalaires, exceptions non précises ...). Aux conclusions s'ajoutent des nouvelles techniques et des astuces architecturales qui sont rendues possibles par l'augmentation du degré d'intégration des puces : par exemple la technique de SRB conjugue les avantages de plusieurs choix sans pénaliser la complexité de la puce et des logiciels.

Les premières versions seront très limitées : une seule instruction décodée par cycle, pas de SRB, petite TLB mixte data/code, pas de FPU, division entière lente et rudimentaire, tout ceci pour faciliter le prototypage et la validation de l'approche avec moins d'un million de transistors.

Superpipeline :

Commençons par décrire les contraintes spécifiques au FC0 : c'est un prototype qui sera pénalisé d'office par le manque de moyens disponibles pour le projet et pourtant il faut jouer dans la cour des processeurs commerciaux actuels. Il faut absolument se découpler des technologies et permettre à n'importe quelle implémentation de fonctionner à la vitesse maximale permise dans chaque cas, afin qu'une amélioration de la technologie ou une rallonge budgétaire se traduise automatiquement par une augmentation de la performance. La solution est de réduire au maximum la longueur des fils et le nombre de transistors à traverser à chaque étage du pipeline. Cette idée est similaire au principe de "superpipeline" mais ici le point de départ est une sorte de "brique de légo" logique qu'il est possible de faire tenir dans environ 10 transistors ou 5 portes logiques de base. Chaque étage remplit donc une fonction la plus simple possible, comme sélectionner un bit ou additionner deux octets. Paradoxalement cela simplifie la conception car il est impossible d'avoir des chemins critiques longs "cachés" par des cas exceptionnels. En conséquence, une instruction prendra un temps variable selon la complexité de l'opération : par exemple, un cycle pour une opération logique, deux pour une addition sur 64 bits.

Scoreboard/Xbar :

Puisque le transfert entre le banc de registres registres (**R7**) et les unités d'exécutions (**EU**) prend un temps important, puisqu'il faut pouvoir gérer des opérations qui prennent un temps variable et puisqu'il faut permettre un passage de résultats anticipé entre les UE, un crossbar (**Xbar**) a été intercalé entre les EU et le R7 afin d'en diminuer le nombre de ports et la charge sur le bus. Le Xbar a un rôle purement de mouvement des données et est une sorte de carrefour pour toutes les informations. Il a donc une certaine complexité irréductible, une occupation importante de la surface de la puce et une place centrale qui prend un cycle dédié dans le déroulement du pipeline général.

Une instruction de type arithmétique sera alors exécutée dans le FC0 avec les étapes suivante, qui prennent un ou plusieurs cycles :

- fetch (lecture du buffer d'instruction)
- décodage de l'opcode
- résolution des dépendances de registres et validation de l'instruction (−> un cycle minimum), lecture des registres sources, validation des pointeurs
- Xbar (peut être bypass si certaines conditions sont reconnues)
- opération (ASU, ROP2, IMU, IDU, LSU, INC...) −> au moins un cycle
- Xbar (la valeur du résultat peut être utilisée directement au cycle suivant)
- R7 : écriture du résultat dans les registres.

Ce déroulement varie pour les autres types d'instructions qui ne font que lire ou écrire un registre. Dans TOUS les cas, les valeurs transitent par le Xbar pour conserver la cohérence du contenu du R7 : des valeur peuvent être "cachées" à certains endroits (comme le scoreboard ou la LSU). Le Xbar est un point de passage obligé pour simplifier l'architecture et réduire la longueur des bus.

Toutes ces opérations sont gérées par un "scoreboard" qui indique quels registes sont prêts ou non, afin d'empêcher l'envoi d'une instruction ayant des dépendances de registres non résolues lors des cycles précédents. Il est donc possible d'envoyer de nombreuses instructions "lentes" à plusieurs EU indépendantes, travaillant sur des registres différents, sans bloquer le processeur. Ceci est particulièrement important pour la LSU car un cache miss bloquerait le programme, alors qu'avec le FC0 il suffit de programmer l'instruction largement en avance pour éviter les pénalités d'accès : le programme ne sera bloqué que si une instruction lit le registre destination d'un "load" alors que la donnée n'est pas prête.

Exceptions :

Si une instruction effectue une opération non valide, il faut pouvoir l'isoler et déclencher le système de gestion d'erreurs de l'OS. Ces erreurs peuvent avoir plusieurs raisons, elles peuvent se produire à divers endroits qu'il faut discerner, mais la plupart des CPU RISC laissent le logiciel gérer ces problèmes complexes. Les "exceptions imprécises" nécessitent une connaissance intime du CPU pour concevoir le logiciel de gestion.LA cela s'ajoute le problème du pipeline particulier ("OOOC" pour "Out Of Order Completion") du FC0 : une instruction peut écrire un résultat dans le R7 *avant* que l'instruction précédente ne fasse de même : cela rend la tâche de l'analyse de l'état du CPU très complexe. La solution employée par les RISC récents est d'utiliser des registres temporaires, un luxe de complexité que nous ne pouvons pas nous payer, car il faut conserver beaucoup d'informations et gérer leur duplication sur la puce.

Le FC0 est parti de l'approche suivante : si les exceptions "précises" nécessitent des mécanismes complexes, c'est à cause des instructions. Certaines instructions sont très simples alors que d'autres impliquent des mécanismes inutilement complexes. La solution au problème est donc de mettre au point les instructions dans le but de garantir que TOUTES les exceptions se produisent à un seul endroit : au niveau du décodeur d'instructions. La gestion est alors très simple (similaire à une instruction de saut) et le pipeline est épuré : non seulement il n'y a pas d'étage supplémentaire dans le pipeline pour temporiser les résultats, mais en plus la gestion des accès à la mémoire est aussi simple que la lecture d'un registre (même en mode paginé). Une astuce consiste à paralléliser des opérations usuellement séquentielles, les registres permettant de garder les résultats temporaires au cas où le flux d'instructions serait interrompu.

Par exemple, dans le F−CPU, l'instruction "load" a été parallélisée en rendant l'opération de chargement proprement dite indépendante du calcul des pointeurs et de leur vérification : elle effectue plusieurs opérations simultanée.

- Elle vérifie si le registre pointeur est valide grâce à une petite cache et quelques *flags*.
- Elle transfère la donnée venant du buffer de mémoire vers le registre destination, en décalant la donnée ou changeant son endian si nécessaire, puis traverse le Xbar.
- En parallèle, le pointeur peut être additionné à un autre registre et le résultat est ensuite vérifié dans la TLB (en suivant le chemin d'une addition normale, et la TLB vérifie le résultat par un port spécial du Xbar).

A la fin de cette séquence, un autre accès à la mémoire peut être effectué avec le même registre pointeur car la TLB aura mis à jour le flag de validité associé au registre pointeur. Si un problème de pointeur apparaissait, il serait détecté au niveau du décodage, et pas au milieu du pipeline : il n'y a pas besoin de sauvegarder tout le pipeline pour poursuivre l'exécution du programme après que l'erreur (le plus souvent une erreur de page) soit traitée.

Cela change radicalement par rapport aux processeur RISC normaux mais aucune instruction n'est émise pour exécution si elle est invalide, ce qui simplifie le matériel. C'est une forme d'évolution du concept RISC original car il va dans le sens de l'adéquation entre le matériel et le logiciel : il faut parfois changer de point de vue pour simplifier le problème. Ici, un petit changement de sémantique des instructions permet de réduire la complexité de l'architecture et d'augmenter sa capacité à s'étendre (par ajout d'unités d'exécution et de décodeurs d'instructions parallèles). Un FC0 superscalaire sera donc moins difficile à concevoir qu'un processeur RISC classique.

3r1w/2r2w :

Certaines opérations utiles en pratique nécessitent plus de 2 lectures de registres ou plus d'une écriture. Or les CPU RISC classiques limitent les instructions à 2 lectures et 1 écriture par cycle, afin de diminuer le nombre de ports du R7 et la surface de silicium.

Dans le cas du FC0, deux instructions ou plus peuvent se terminer au même cycle : il faut donc pouvoir écrire deux résultats à la fois, ce qui lève la limitation classique.

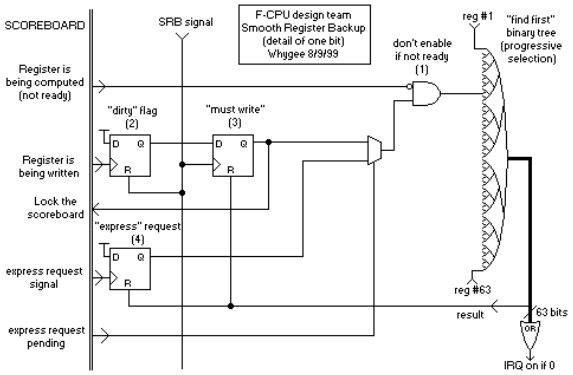
De plus, il faut parfois pouvoir lire 3 registres en même temps, par exemple pour l'instruction MAC mais surtout STORE avec mise à jour de pointeur : il faut lire le pointeur, l'additionner à un autre registre, tout en lisant un troisième pour l'envoyer vers la mémoire.

Le FC0 en version 1 instruction/cycle a donc un R7 avec 3 ports de lecture et 2 ports d'écriture. L'allocation des ports d'écriture est gérée au décodage de l'instruction, et un cycle d'attente est introduit si plus de 2 registres doivent être écrits à la fois.

Le jeu d'instruction profite de la levée de cette limitation en apportant des instructions plus riches et plus adaptées à certains algorithmes (MAC par exemple). Lorsqu'il faut écrire deux registres à la fois, et comme un seul champ est réservé à cette fonction dans l'opcode, le registe "voisin" est écrit (sauf dans le cas d'un LOAD).

SRB :

Lors d'un changement de tâche ou d'une interruption, il faut sauver 63 registres puis en charger d'autres, ce qui prend beaucoup de temps et de bande passante mémoire. La technique du SRB (Smooth Register Backup) réduit le temps de latence de cette opération en entrelaçant la sauvegarde du contexte de l'ancienne tâche avec les instructions de la "nouvelle" tâche, ceci "à la demande", en fonction des instructions exécutées par la nouvelle tâche. D'une part le temps de réaction lors d'interruptions est diminué, d'autre part la bande passante vers la mémoire centrale est mieux gérée, et il n'y a pas besoin d'une longue suite d'instructions pour sauvegarder le R7 (ce qui prendrait 63*4=256 octets de code en mémoire) quel que soit le nombre de registres. Malgré ses 64 registres, un FC0 doté du SRB est adapté aux contraintes de temps réel sans pour autant utiliser plusieurs bancs de regsitres comme dans le SPARC (ce qui permet en retour d'avoir plus de registres). Le mécanisme de SRB permet aussi de créer des instructions de sauvegarde par bloc des registres, comme lors des appels et retours de fonctions. Enfin, le mécanisme de SRB est "sûr", atomique et ne génère pas d'exception : il n'est pas "dangereux" comme une instruction microcodée dans un processeur CISC.



Le SRB n'est pas une unité mais un comportement d'ensemble de plusieurs unités cadencées par le scoreboard et les instructions de la nouvelle tâche. L'idée est d'entrelacer la sauvegarde des registres de l'ancienne tâche avec l'exécution de la nouvelle. L'ordre de sauvegarde est déterminé en partie par le programme qui prend le contrôle du processeur : la LSU (R/W mémoire) "intercepte" sur le Xbar les valeurs à sauver. En effet, chaque instruction effectue la lecture de 3 registres dont un qui écrira sur un registre potentiellement non sauvé. Chaque registre du R7 est associé à plusieurs bits qui indiquent par exemple si le registre a été sauvé ou s'il doit l'être. Un arbre de sélection linéaire progressive permet de sauver les registres dans l'ordre croissant de leur numéro et un mécanisme ("express") est prévu pour changer cet ordre par défaut, en fonction des instructions qui arrivent au même moment.

Programmation :

GCC est en cours de portage sur le F−CPU décrit ici (fichiers hébergés par sourceforge). Cependant, les performances escomptées sont faibles à cause de l'architecture particulière du F−CPU. GCC a été conçu pour les ordinateurs des années 80, pas pour un processeur superpipeline du troisième millénaire, et peu de personnes veulent/peuvent coder en assembleur.

Le projet GNL se déroule en parallèle et devrait permettre le portage, le codage et l'optimisation rapides et faciles de programmes pour de nombreuses plateformes. "GNL is Not a Langage" car il représente les programmes sous forme de graphes, ce qui diminue les fautes de frappe et permet une meilleure interactivité entre le compilateur et le programmeur, surtout lors de la conception et du débuggage. Un front−end C (non ANSI, basé sur le PPCM) et une interface graphique sont en cours de développement. A terme le but est de remplacer les langages textuels dans de nombreux domaines et de favoriser de meilleures pratiques de codage. Ce projet indépendant n'est pas destiné spécialement au F−CPU mais lui permettra de monter rapidement en puissance et facilitera la migration vers la nouvelle plateforme.

Fabriquer un F−CPU :

Bien que les termes exacts de la licence F−CPU ne soient pas encore précisément définis, l'idée principale du projet est de laisser une liberté totale aux entreprises qui fabriqueront (physiquement) les puces à partir des définitions du groupe F−CPU, à condition de

- redistribuer toutes les modifications qui ont été effectuées sur les fichiers originaux
- redistribuer tous les nouveaux fichiers créés pour le F−CPU sous la licence F−CPU
- maintenir une base d'informations, accessible librement et anonymement sur Inernet par FTP ou HTTP, sur toutes les versions des circuits fabriqués (jusqu'à dissolution de la société).

Le but est de garantir la longévité du projet et de l'architecture F−CPU

- en assurant un support "passif" (pas de hotline ou d'investissement lourd en personnel par exemple) et à vie des produits par le fabricant, afin que de vielles machines puissent toujours servir. Il est toujours tentant de couper le support d'un produit pour encourager l'achat d'une nouvelle version : c'est une pratique déloyale envers le consommateur, particulier ou industriel, qui investit dans une architecture.
- en empêchant une vampirisation de la Propriété Intellectuelle par des pratiques anticoncurrentielles, en assurant que toutes les informations sur les produits sont disponibles, à jour et accessibles sans discrimination.

Une fois que le circuit implémentant le F−CPU est fabriqué, l'utilisateur comme le fondeur ont la liberté totale sur le produit, en restant dans la légalité bien entendu, de faire ce qu'ils en veulent : vendre trop cher ou décorer leur salon avec, selon le goût de chacun. Le jeu de la concurrence loyale est chargé d'équilibrer le marché. Naturellement, comme c'est le cas de la GPL, il est possible de vendre des *supports informatiques* contenant la Propriété Intellectuelle ou les travaux dérivés mais en aucun cas ils ne peuvent être vendus eux−mêmes (par exemple *vendus* sur Internet). Une autre mesure est donc imposée pour garantir la liberté des données : elles doivent être disponibles gratuitement sur Internet (par exemple). Une entreprise de distribution ne peut pas vendre l'IP mais que des services associés (diffusion, distribution, sélection, paquetage, vérification, garantie), ce qui ne doit pas l'empêcher de participer à l'effort de développement en fonction des bénéfices.

Conclusion :

Le projet F−CPU était au départ une utopie avouée et un peu maladroite. Son esprit libertaire a cependant catalysé les efforts de nombreux hobbyistes et passionnés qui se sont rassemblés avec succès lors d'opérations ponctuelles, comme lors des réunions ou du dépôt de la marque à l'INPI. Il est incontestable que ce projet, comme les initiatives similaires, a un très fort potentiel car il correspond aux besoins de la plupart des utilisateurs d'ordinateurs.

Les efforts nécessaires pour créer un microprocesseur fonctionnel sont considérables et bien qu'une grande partie des bases de l'architecture soient posées, les détails techniques sont en mesure de freiner le développement dans un groupe informel. Le groupe F−CPU n'est donc qu'au début de sa structuration (fonctionnelle et légale).

Le premier prototype tarde à apparaître car de nombreux détails sont encore indéterminés : plus qu'une puce, c'est tout un monde qu'il s'agit de recréer et les membres manquent de temps pour se consacrer au projet. Le développement est donc très lent et frustrant pour les personnes qui désirent un F−CPU tout de suite, mais il est nécessaire de prendre toutes les précautions utiles quand on définit une architecture destinée à durer *très* longtemps : un mauvais jugement peut handicaper le projet dans le futur et il faut être très prudent. Les premières puces attendront le temps qu'il faudra pour qu'elles ne souffrent pas d'erreurs de jeunesse qui pourraient anéantir les espoirs qu'elles portent. Tant qu'à être libre, autant faire du bon boulot.