Cahier des charges

Version du 20 mars 2007

Encadrant :
Frédéric Mallet

Etudiants :
Benjamin Baudouin
Olivier Orabona (*chef de projet*)
Laurent Rodriguez

Nom du projet :
FUCS
Flexible Universal CPU Simulator
(http://fucs.free.fr)

Table des Matières

1 Introduction

1.1 Résumé et objectifs

Il s'agit de réaliser un prototype de simulation de processeur au niveau registres (Register Transfer Logic –RTL) permettant d'intervenir sur différents aspects – unités logiques – de l'architecture interne du processeur : lecture des instructions, décodage, exécution, observation du pipeline, algorithme de prédiction de branchement. Il n'est pas question ici de virtualiser une machine complète afin d'exécuter un système d'exploitation invité.

Le but est de simuler le comportement du processeur en décomposant son fonctionnement en unités logiques. C'est pour cette raison que dans un premier temps, nous nous consacrerons à l'étude et la modélisation du fonctionnement d'un processeur, couramment utilisé sur un ordinateur personnel, l'Intel x86. Puis, dans un second temps, nous tâcherons de concevoir une application qui permettra de contrôler l'exécution de programmes tests « simples » et un plus complexe : une implémentation du célèbre algorithme de chiffrement à clé publique : RSA.

Notre modélisation s'attachera à pouvoir représenter tous les composantes internes du processeur tout en se focalisant, en premier lieu, sur les plus importants : les registres, la mémoire, le pipeline, le décodage d'instructions (désassemblage), la conversion en micro-opérations RISC et enfin à l'unité de prédiction de branche.

Les applications possibles pour ce type de programme sont très variées :

- Concevoir une architecture d'ordinateurs
- Vérifier le code généré par un compilateur
- Déboguer un programme
- Faire des optimisations
- Faire de l'analyse de performances sur tel ou tel processeur
- Enseigner le fonctionnement d'un processeur

1.2 Livrables

Au final, une application exécutable tournant sous GNU/Linux, permetta d'utiliser une bibliothèque de simulation. Cette application aura une interface graphique. Nous fournirons aussi une série d'implémentations de programmes tests à fin d'exemples démontrant l'utilisabilité de l'application, une documentation utilisateur, une description de l'architecture d'un processeur de type x86 (Intel). Enfin nous fournirons toutes les bibliothèques permettant de simuler cette architecture, ces bibliothèques représenteront les unités logiques internes au processeur que nous aurons modélisé.

1.3 Études déjà effectuées

Des études sur ce domaine ont déjà été effectuées. Ainsi le document intitulé « THE MICROARCHITECTURE OF INTEL AND AMD CPU'S: AN OPTIMIZATION GUIDE FOR ASSEMBLY PROGRAMMERS AND COMPILER MAKERS » [1] , offre un panel de découvertes expérimentales faites sur différentes générations de processeurs x86 (Intel et AMD).

Des articles publiés sur Internet se focalisent sur l'aspect architecture des ordinateurs x86 et les potentielles failles qu'elles développeraient intrinsèquement. Ainsi les articles « ON THE POWER OF SIMPLE BRANCH PREDICTION ANALYSIS »[2] a eu beaucoup d'impact dans la communauté des crypto-analystes car elle envisage de pouvoir « casser » l'algorithme RSA sans autre outil qu'un ordinateur personnel et un programme bien conçu.

Suite à l'annonce de cette nouvelle forme d'attaque contre l'algorithme RSA des articles ont répondu à cette théorie. Citons au passage deux d'entre eux : « PEUT-ON CASSER UNE CLÉ DE CRYPTAGE EN QUELQUES MILLISECONDES ? »[3] et un article publié dans un blog [4].

1.4 Études menées sur des sujets voisins

- **Wine (http://www.winehq.com)**

Wine est une interface entre l'API Windows et le sous système graphique X et le système d'exploitation GNU/Linux. En ce sens on peut dire que Wine émule le système Windows sans pour autant émuler tout l'infrastructure. En effet, toutes les bibliothèques graphiques Windows ont été réécrites et en ce sens il ne constitue par une violation de la propriété intellectuelle. Il permet de pouvoir porter des programmes Windows sous Linux et de pouvoir exécuter ces derniers sans aucune modification.

- **VMWare (http://www.vmware.com)**

VMWare créé un environnement clos permettant d'émuler une machine complète. Cela permet d'installer n'importe quel système d'exploitation compatible avec la plateforme sur laquelle VMWare s'exécute sans que le système d'exploitation "invité" ne puisse distinguer d'une machine physique. C'est ce que l'on appelle une virtualisation (voir définition plus bas).

- **Bochs (http://bochs.sf.net)**

Bochs est un émulateur de machines x86. A l'instar de VMWare il permet d'installer un système d'exploitation dans un environnement clos mais du fait de sa portabilité, il doit simuler tout le jeu d'instructions du x86. Il est donc instrinsèquement plus lent qu'un logiciel de virtualisation tel que VMWare.

---

1 http://www.agner.org/optimize/microarchitecture.pdf

2 (Cryptology ePrint Archive, Report 2006/351) http://eprint.iacr.org/2006/351.pdf

3 http://www.futura-sciences.com/news-peut-on-casser-cle-cryptage-quelques-millisecondes_10064.php

4 http://sid.rstack.org/blog/index.php/2006/11/22/152-la-fin-du-monde

- **QEMU (http://www.qemu.org)**

Qemu est comparable à Bochs d'un point de vue simulation de toute une machine. Une machine type avec des composants qui n'existent pas forcément en hardware et sont donc émulés. Il peut néanmoins profiter de certaines accélérations matérielles grâce à son module kernel ce qui lui permet de se rapprocher des performances d'un logiciel tel VMWare.

- **Simple Scalar (http://www.simplescalar.com/)**

Simple Scalar est un simulateur de jeu d'instructions. C'est un logiciel non libre au sens GNU. « The SimpleScalar tool set is a system software infrastructure used to build modeling applications for program performance analysis, detailed microarchitectural modeling, and hardware-software co-verification. Using the SimpleScalar tools, users can build modeling applications that simulate real programs running on a range of modern processors and systems. The tool set includes sample simulators ranging from a fast functional simulator to a detailed, dynamically scheduled processor model that supports non-blocking caches, speculative execution, and state-of-the-art branch prediction. »

- **SESC (http://sesc.sf.net)**

Tout comme Simple Scalar, SESC est un simulateur d'architecture d'ordinateur. Ils ont les mêmes fonctionnalités, celui ci étant en plus avec une licence compatible OSI. Il ajoute en outre la possibilité de simuler d'autres environnements comme les processeurs CMP et PIM. « SESC is a cycle accurate architectural simulator. It models a very wide set of architectures: single processors, CMPs, PIMs, and thread level speculation. »

1.5 Définitions et acronymes

Voici une liste des définitions et leurs acronymes associés tels qu'ils seront utilisés par la suite dans ce document. Les définitions ci dessous ont été tirées de sites internet tels que Wikipedia.

- **RISC (Reduced Instruction Set Computer) :**

L'architecture RISC fait le choix de limiter le jeu d'instructions à seulement quelques unes, imposant, en contrepartie, à toutes, un nombre identique de cycles pour s'exécuter. De cette manière, il est possible de débuter une nouvelle instruction à chaque cycle d'horloge: ceci constitue le « pipeline ». L'avantage de cette technique est que désormais, le processeur se comporte comme s'il y avait une instruction exécutée par cycle d'horloge. De plus, la division de chaque instruction en plusieurs étapes autorise une fréquence d'horloge plus grande puisque la profondeur combinatoire entre deux registres est diminuée. Ces deux caractéristiques ont pour conséquence une division du temps d'exécution pour toutes les instructions de base.

Cela se paye au prix d'une certaine diminution de lisibilité du code gênante si l'on programme en assembleur et surtout si on l'optimise. En temps d'exécution, le code C optimisé se montrait en général plus performant en vitesse pure grâce à des astuces d'usage de l'effet pipeline par le compilateur.

• CISC (Complex Instruction Set Computer)

Au contraire du RISC, le CISC possède un jeu d'instructions important et mélange les modes d'adressages complexes : adressage basé et/ou indéxé et/ou avec déplacement. Les avantages sont nombreux, l'empreinte mémoire du code est beaucoup plus dense ce qui est intéressant pour minimiser la taille du cache instruction. La possibilité de microprogrammation, donc de corriger le jeu d'instructions (cela peut être utile pour corriger des bugs). Enfin il permet d'utiliser des instructions très complexes non (ou mal) gérées par les compilateurs mais très rapides (ex: instructions SIMD).

Cependant, le microprocesseur est plus compliqué à accélérer (problème pour pipeliner le moteur d'exécution), il est globalement plus complexe à concevoir qu'un processeur RISC et les compilateurs ont des difficultés à générer des instructions complexes.

• Micro opérations

Du fait des possibilités d'optimisation offertes par l'architecture RISC, certaines générations de processeur CISC fonctionnent sur deux niveaux. Ils fournissent toujours une ensemble complexe d'instructions de « haut niveau » mais les implémentent en interne comme une séquence d'instructions plus simples les micro opérations qui ressemblent au jeu d'instructions RISC. Quand le processeur doit décoder une macroinstruction CISC, il l'a décode en micro opérations RISC.

• Instruction pipeline

Une « instruction pipeline » est une technique utilisée dans la conception d'ordinateurs pour augmenter leurs performances. A chaque tic d'horloge le processeur fait en sorte que les instructions « avancent » dans le pipeline. Le pipeline est matérialisé par des unités logiques internes qui se chargent de chaque étape :

- *Instruction Fetch* : lit l'instruction depuis la mémoire
- *Instruction decode and register fetch* : décode l'instruction et récupère les données des registres
- *Execute* : exécute à proprement parler l'instruction
- *Memory access* : si un accès mémoire doit avoir lieu alors il se fera à cette étape
- *Register write-back* : si une mise à jour des registres est nécessaire elle se fera à cette étape

Lorsqu'une instruction a été lue, elle passe dans l'unité de décodage et au même moment l'instruction suivante est lue. Au troisième tic d'horloge, la première instruction sera exécutée, la deuxième sera décodée et une nouvelle instruction sera lue. Et ainsi de suite ...

- ## Prédiction de branchement

Il s'agit d'une unité logique présente dans tous les processeurs modernes qui agit au niveau du pipeline. En effet lorsqu'un branchement conditionnel à lieu, le processeur doit savoir quelle sera l'instruction suivante à exécuter. Or la condition du branchement n'a peut être pas été encore calculée. Comme le processeur ne peut pas être sûr de l'instruction suivante, il doit alors procéder de manière probabiliste. C'est à ce moment là qu'intervient l'unité de prédiction de branchement. Elle devra dire au processeur si tel branchement a un probabilité « raisonnable » d'être pris ou de ne pas être pris. Si l'unité de branchement a vu juste, le processeur pourra continuer à lire son flot d'instructions après le branchement et le pipeline ne sera pas cassé. Par contre si le branchement ne s'est pas passé comme prévu, alors le processeur devra vider le pipeline en cours car les instructions lues par la suite ne sont pas valides.

- ## Simulation

Le but d'une simulation est de pouvoir modéliser un concept abstrait. Ici nous simulerons un jeu d'instructions pour un processeur donné. Nous mimerons le comportement du processeur.

- ## Emulation

Un émulateur a pour but d'imiter le comportement de toute une architecture matérielle de telle manière que tout programme communicant avec le matériel émulé pensera qu'il s'agit du vrai matériel avec lequel il dialogue. L'architecture émulée est intrinsèquement différente de l'architecture hôte.

- ## Virtualisation

La virtualisation est une émulation de toute ou partie du matériel mais est prévu pour la même architecture à la fois pour la machine « émulée » que pour celle qui « émule ». De la sorte, la machine émulée peut tirer parti des avantages de la même architecture sans qu'il soit nécessaire pour le programme de simuler un processeur différent. Les performances en sont donc largement accrues.

- ## Formats de fichiers

Tous les fichiers exécutables ont un format bien spécifique, dépendant du système d'exploitation, qui permet à ce dernier de faire les opérations nécessaire en interne afin de charger le code exécutable du fichier et de lancer son exécution dans l'espace d'adressage qui lui est réservé. Sous Linux, le format de fichiers exécutables est ELF (Executable and Linking Format). Sous Windows il s'agit du format PE (Portable Executable). Certains systèmes n'ont pas besoin de format évolué et se contentent de mélanger code et données dans un même segment d'adressage et l'absence de format fait a un nom : le format « raw » (brut).

- ## Extreme Programming

Technique de programmation dans lequel les tests unitaires et les tests

d'intégration s'éffectuent tout au long de la production de code. Il est très bien adapté au développement en binôme. Chacun peut alors se consacrer au développement du logiciel pendant que l'autre s'occupe des tests et vice-versa. La documentation y joue un grand rôle ainsi que le style de programmation car on ne doit pas perdre de temps à comprendre ce que l'autre a fait.

2 Organisation du projet

2.1 Processus

Compte tenu des prérequis nécessaires à l'aboutissement du projet, celui ci se décomposera en trois phases. La première est une phase de mise à niveau dans les langages de programmation qui seront utilisés à savoir le C++ et l'assembleur x86, voir ou revoir le fonctionnement de l'algorithme RSA, effectuer une mise au point sur ses connaissances en conception UML 2.0 et enfin l'apprentissage de la bibliothèque graphique wxWidgets. Cette première phase commence le 15 mars pour finir le 30 mars. A la fin de cette période, des programmes tests écrits en assembleur serviront à la fois d'exemples et comme moyen de contrôle.

Dans un second temps, il sera important de modéliser notre architecture logicielle. C'est une étape importante à faire avant toute implémentation. Une interface graphique sera conçue et un processeur à simuler sera choisi à ce moment là. Des documents de synthèse seront produits à l'issue de cette seconde phase qui s'étalera de la semaine du 9 avril à la semaine du 23 avril.

Enfin pendant toute la durée restante du projet, la troisième et dernière phase consistera à implémenter une interface graphique et le moteur de simulation qui aboutira au final à l'application.

2.2 Organisation structurelle

Sachant les contraintes temporelles et le volume horaire que chacun pourra allouer au projet, l'organisation s'adaptera au fur et à mesure des difficultés rencontrées. Un planning particulier a été conçu pour Benjamin Baudouin, étudiant salarié.

2.3 Limites et interfaces

Dans cette première version du simulateur, nous nous limiterons dans les fonctionnalités. Ainsi, nous ne traiterons pas les programmes qui font des appels aux API. La programmation s'effectuera dans un environnement qui est famillier pour nous, à savoir le système GNU/Linux. Nous nous limiterons aussi à simuler une seule famille de processeurs ainsi qu'un jeu d'instructions réduit au strict minimum pour faire tourner nos programmes de tests ainsi que l'algorithme RSA.

3 Gestion

3.1 Objectifs et priorités

(1) Bien appréhender les prérequis nécessaires en C++, assembleur, UML et RSA

(2) Concevoir des programmes tests simples

(3) Concevoir une interface graphique avec wxWidgets

(4) Implémenter l'algorithme RSA en assembleur et en déduire le jeu d'instructions nécessaire à l'exécution de cet algorithme.

(5) Modéliser un processeur

(6) Modéliser le moteur de simulation

(7) Créer un framework de conception de composants (unités logiques)

(8) Charger des programmes « raw »

(9) Implémenter les fonctions de base d'exécution

(10)Faire des tests d'intégration

3.2 Hypothèses, dépendances, contraintes

Durant ce projet, nous aurons à définir plusieurs hypothèses de travail:

- licence d'utilisation (à déterminer en fonction des bibliothèques utilisées)
- choix d'un processeur x86

Les dépendances seront surtout au niveau de l'interface graphique à cause du sous sytème de fenêtrage, avec la bibliothèque wxWidgets v2.8.2 (http://www.wxwidgets.org).

Enfin, nous aurons à la fois comme contrainte et comme limite implicite l'architecture processeur 32 bits.

3.3 Gestion du risque

Nous avons identifié trois types de risques :

- ➔ risque dans l'approfondissement des connaissances en C++, UML
- ➔ risque dans l'apprentissage du langage d'assemblage, de l'architecture des ordinateurs et de l'IHM (par une nouvelle bibliothèque wxWidgets)
- ➔ risque des délais par le fait de la présence d'un étudiant salarié pour lequel nous définirons un calendrier adapté

Pour les limiter au maximum, nous avons déjà et nous prévoyons de rencontrer l'encadrant au moins une fois par semaine. L'équipe ne peut pas se voir ensemble très régulièrement à cause des contraintes propres à chaque membre. Ce qui nous amène à prendre les décisions suivantes afin de parer aux éventuelles difficultés :

- Des rencontres avec le chef de projet
- Un site wiki mis en place sur http://fucs.free.fr
- Une réévaluation régulière du planning

3.4 Moyens de contrôle

Les moyens de contrôles que nous avons décidé de mettre en place font appel à la base à des techniques issues de l' « Extreme Programming ». En effet celui ci met en avant le développement conjoint d'une application et de ses tests. Les résultats de la simulation devront refléter les résultats attendus par l'exécution dans un environnement réel des programmes de test. La comparaison avec des résultats fournis par une bibliothèque de référence (OpenSSL) permettront de valider l'algorithme que nous mettrons en place. Enfin, nous disposons d'études déjà effectuées (cf. plus haut dans ce document) afin de nous guider dans la conception des composantes internes au processeur.

4 Technique

4.1 Méthodes et outils employés

- ➔ C++ (suite GCC) version 4.1.2
- ➔ comme dit précédemment « Extreme Programming » : programmes de tests, validation par des implémentations (re)connues comme « sûres »
- ➔ Assembleur (NASM) version 0.98.39
- ➔ UML
- ➔ IHM avec wxWidget version 2.8.2
- ➔ Doxygen version 1.5.1 pour générer la documentation

4.2 Documentation

La documentation aura une place prépondérante dans notre projet de par la nécessité de communication avec la méthode d' « Extreme Programming ». La documentation prendra la forme de documents électroniques au format Portable Document Format (PDF) pour tout ce qui est synthèse de documents. La documentation de code se fera par le logiciel sous licence GPL, Doxygen.

5 Calendrier

6 Fonctions du produit

6.1 Globales

- ➔ Interface graphique facile à appréhender pour tous les utilisateurs

- ➔ Chargement de programmes « raw »

- ➔ Visualisation du code désassemblé, des registres du processeur et de la mémoire au cours de la simulation
- ➔ [optionnel] Chargement de programmes au format PE ou ELF
- ➔ [optionnel] Chargement et sauvegarde d'un projet (programme simulé)

6.2 Pour le débogage

- ➔ Exécution pas à pas
- ➔ (Re-)lancer le programme
- ➔ Interrompre l'exécution
- ➔ Reprendre l'exécution
- ➔ [optionnel] Retour arrière dans le flot d'instructions
- ➔ [optionnel] Sauvegarde mémoire et contexte d'exécution
- ➔ [optionnel] Gestion des points d'arrêts dans le programme

6.3 Pour la conception d'architecture matérielle

- ➔ Description d'un processeur
- ➔ Chargement de composants internes (bibliothèques)
- ➔ [optionnel] Visualisation du fonctionnement des différents composants
- ➔ [optionnel] Vue des micro-opérations

6.4 Pour la validation de code

- ➔ [optionnel] Décompilation du code et des structures dans un pseudo langage
- ➔ [optionnel] Heuristiques de détection de code

6.5 Pour l'optimisation

- ➔ Compteur de temps d'exécution en cycles d'horloge
- ➔ [optionnel] Aperçu de l’empreinte mémoire (espace) et du temps d’exécution

6.6 Pour l'analyse de performances

- ➔ [optionnel] Charger plusieurs définitions de processeurs

➔ [optionnel] Afficher un tableau comparatif en temps d'exécution (cycles d'horloge)

6.7 Pour l'enseignement

➔ [optionnel] Aide contextuelle

➔ [optionnel] Schémas graphiques représentatifs du fonctionnement des différents composants

7 Contraintes non fonctionnelles

Ces contraintes non fonctionnelles sont pour le moment à l'état d'hypothèses. Nous nous laissons jusqu'à la fin de la phase d'étude pour affiner ces paramètres.

7.1 Plateforme matérielle

La plateforme matérielle choisie est, de part sa nature répandue, l'Intel x86.

7.2 Temps de réponse

Le temps de réponse doit être linéaire par rapport au temps d'exécution effectif.

7.3 Empreinte mémoire

Aucune contrainte sur l'empreinte mémoire. Cela dit pour ce type d'applications, une programmation efficace implique de facto des contraintes au niveau de la qualité de programmation. Donc des algorithmes optimum là où c'est nécessaire.

8 Annexes

Suites prévues

- extension de l'implémentation pour d'autres processeurs
- implémentation possible de la simulation des appels API
- exploration d'architecture autre que CISC
- registres et instructions étendus (FPU, MMX, SSE, …) (à définir)

Parties concernées par le déroulement du projet et ses résultats

Toutes les personnes désireuses de modéliser, tester, déboguer, optimiser des processeurs et des programmes. Du cours d'architecture système aux laboratoires de recherche. Le projet pourra être visible à la communauté Open Source sur des sites spécialisés tel que SourceForge avec une licence compatible OSI.

Phase d'étude : (9 avril -

- étudier l'algorithme RSA
- étudier le fonctionnement de l'architecture x86
- choisir une version du x86 dont on détaillera le fonctionnement
- réaliser un travail de synthèse des documents qui sont à notre disposition
- délivrables : implémentation de référence de RSA, document de synthèse sur le processeur choisi, dessin de l'interface prévue (à réfléchir en fonction du temps)

Phase de conception :

- modéliser un processeur de type x86
- concevoir une bibliothèque extensible et flexible pour concrétiser la modélisation
- concevoir une IHM répondant aux critères de simplicité et de complétude des fonctionnalités qui seront proposées
- concevoir une série de programmes tests simples permettant de faire des *test suites*
- concevoir une version assembleur de l'algorithme RSA
- délivrables : simulateur (source, binaires), programme assembleur (RSA), tests suites, documentation (utilisateur, conception, développement d'extensions).

Planning (plus de details de qui et quand)

Phase de mise à niveau : (15 mars - 30 mars)

- asssembleur
- C++
- UML
- Apprentissage de wxWidgets
- délivrables : premiers programmes simples en assembleur