UNIVERSITÉ BORDEAUX 1 Sciences Technologies

Année 2009-2010
Étape : Licence Sciences Technologies (semestre 4)
UE : INF155 (Architecture des ordinateurs)
Modalités : Épreuve sans document

Date : 11 mai 2009
Heure : 14h00
Durée : 3h00

1 Modèles d'architecture (question de cours)

Expliquez, en une ou deux phrases, le principe que désigne chacun de ces termes :

1. architecture fondée sur un *pipeline* ;
2. processeur *superscalaire* ;
3. processeur doté de la technologie *Intel® HyperThreading* ;
4. processeur multi-cœur.

2 Circuits combinatoires

On souhaite construire un circuit «`XOR3`» à 3 entrées et 1 sortie réalisant la fonction «*ou-exclusif*» : la sortie S de ce circuit vaut 1 si l'une et une seule de ses entrées vaut 1, sinon la sortie vaut 0.

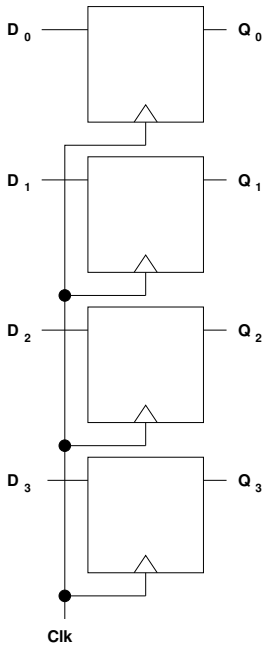
Q1 Donnez une implémentation de ce circuit à l'aide de portes logiques simples.

Q2 Pour quelles valeurs des entrées (a, b et c) le circuit `XOR3` donne-t-il un résultat différent en comparaison avec un circuit `XORoXOR` qui réaliserait $(a \oplus b) \oplus c$?

Q3 On dispose d'un additionneur (sur 1 bit) à 3 entrées (x, y et z) et deux sorties (s la somme et c la retenue). Construisez un circuit `XOR3` utilisant un tel additionneur.

3 Circuits séquentiels

Le circuit ci-contre est un registre de 4 bits conçu en juxtaposant simplement des «bistables D». On souhaite étendre ce circuit pour pouvoir l'utiliser dans la version pipeline du processeur Y86, et plus précisément pour stocker la valeur `D_icode` (code de l'instruction en cours à l'étage *Decode*).



Q1 Dans sa version actuelle, le registre prend la valeur (D_3, D_2, D_1, D_0) au début de chaque cycle. Dans certains cas, il est toutefois nécessaire de forcer une instruction à demeurer à l'étage *Decode* au cycle suivant.
On veut donc ajouter un fil de commande `stall` permettant, lorsqu'il vaut 1, de forcer le registre à conserver sa valeur actuelle au prochain cycle. Ajoutez au circuit ci-contre les portes logiques et les connexions nécessaires à ce fonctionnement.

Q2 Parfois, il est également nécessaire d'injecter une «*bulle*» à l'étage *Decode* du pipeline. Sachant que le code de l'instruction `NOP` est 0, ajoutez un fil de commande `bubble` au circuit pour déclencher, lorsqu'il vaut 1, l'injection d'une bulle au cycle suivant.

4 Appel de fonction indirect

Le jeu d'instruction Y86 ne possède pas d'instruction permettant d'appeler une fonction dont l'adresse est contenue dans un registre. C'est pourtant fort utile. Pour contourner le problème, on se propose d'écrire une fonction `indirect_call` dont le prototype en langage C serait (les «...» désignant un nombre variable d'arguments) :

```
int indirect_call(int (*func)(...), ...);
```

En d'autres termes, pour appeler une fonction dont l'adresse est contenue dans le registre `eax` avec deux arguments respectivement contenus dans `esi` et `edi`, on écrirait le code ci-contre.

```
pushl %edi   // arg 2
pushl %esi   // arg 1
pushl %eax   // func
call indirect_call
...
```

Q1 Pour réaliser la fonction `indirect_call`, l'idée est d'utiliser l'instruction `ret` après avoir réarrangé la pile de manière à provoquer un retour vers... l'adresse `func`. Donnez le code de la fonction `indirect_call`. Attention à bien garantir que le retour de la fonction `func` se passe bien.
NB : les registres *caller-save* du Y86 sont `eax`, `ecx` et `edx`.

Q2 Combien de paramètres faut-il dépiler, au retour de l'appel de `indirect_call` (i.e. «...» dans le code ci-contre), pour nettoyer la pile ? Expliquez.

5 Version pipelinée du processeur Y86

Q1 Examinez le programme ci-contre. Que fait-il ? Chronogramme à l'appui, indiquez combien s'écoulent de cycles entre l'instant où l'instruction `rmmovl %eax, (%edi)` entre dans le processeur pour la première fois et l'instant où l'instruction `halt` entre à son tour dans le processeur. Attention à la prédiction de branchement...

```
.pos  0
      irmovl t,    %edi
      mrmovl size, %ecx
      xorl   %eax, %eax
      irmovl 1,    %ebx
      irmovl 4,    %edx
boucle:
      rmmovl %eax, (%edi)
      addl   %edx, %edi
      subl   %ebx, %ecx
      jne    boucle
      halt
.pos  100
size: .long  2
t:
```

Q2 On se propose d'introduire une nouvelle instruction dans le processeur Y86 : `STOSWL` (*Store String Word*). Cette instruction (sans argument) effectue l'équivalent de `rmmovl %eax, (%edi)`, tout en ajoutant 4 au registre `edi` (la valeur de `edi` utilisée par `rmmovl` est celle avant incrémentation).
La feuille jointe au sujet reproduit l'essentiel du fichier pipe-std.hcl, qui décrit la version pipelinée du processeur Y86. Ecrivez directement sur cette feuille les ajouts que vous proposez pour traiter cette nouvelle instruction (de code `STOSWL`).
NB : n'oubliez pas d'inscrire votre numéro d'anonymat sur la feuille avant de l'insérer dans votre copie.

Q3 Quel est le principal intérêt d'utiliser une telle instruction ? Dans le programme ci-contre, si on remplace les deux premières instructions de la boucle par `stoswl`, gagne-t-on des cycles ?

Q4 Pour continuer à réduire le nombre d'instructions au sein de la boucle, on peut s'inspirer des processeurs x86 et envisager d'ajouter une instruction `loop <label>` à notre processeur. Cette instruction décrémente le registre `ecx`, puis réalise un saut à l'adresse `label` si la nouvelle valeur d'`ecx` est différente de zéro.

Comme pour l'instruction `STOSWL`, ajoutez le code `LOOP` partout où c'est nécessaire sur la feuille jointe afin de gérer correctement le fonctionnement de cette instruction. Laissez de côté les aspects relatifs à la prédiction de branchement pour cette question...

Q5 La gestion des mauvaises prédictions de branchement pose problème avec l'instruction `loop`. Contrairement aux instructions de saut conditionnel (`JXX`), l'instruction `loop` appuie sa décision non pas sur des codes de conditions positionnés par des opérations antérieures, mais par le résultat sortant de l'ALU au moment où l'instruction `loop` se trouve à l'étage *Execute*. Expliquez précisément d'où vient le problème et comment on pourrait le corriger (on ne demande pas de le faire sur la feuille annexe).

— Feuille à détacher et à rendre avec la copie —

Numéro d'anonymat :

```
################ Fetch Stage     ###################################

## What address should instruction be fetched at
int f_pc = [
# Mispredicted branch.  Fetch at incremented PC
M_icode == JXX && !M_Bch : M_valA;
# Completion of RET instruction.
W_icode == RET : W_valM;
# Default: Use predicted value of PC
1 : F_predPC;
];

# Predict next value of PC
int new_F_predPC = [
f_icode in { JXX, CALL } : f_valC;
1 : f_valP;
];

################ Decode Stage ######################################

## What register should be used as the A source?
int new_E_srcA = [
D_icode in { RRMOVL, RMMOVL, OPL, PUSHL  } : D_rA;
D_icode in { POPL, RET } : RESP;
1 : RNONE; # Don't need register
];

## What register should be used as the B source?
int new_E_srcB = [
D_icode in { OPL, RMMOVL, MRMOVL  } : D_rB;
D_icode in { PUSHL, POPL, CALL, RET } : RESP;
1 : RNONE;  # Don't need register
];

## What register should be used as the E destination?
int new_E_dstE = [
D_icode in { RRMOVL, IRMOVL, OPL} : D_rB;
D_icode in { PUSHL, POPL, CALL, RET } : RESP;
1 : RNONE;  # Don't need register
];

## What register should be used as the M destination?
int new_E_dstM = [
D_icode in { MRMOVL, POPL } : D_rA;
1 : RNONE;  # Don't need register
];

## What should be the A value?
## Forward into decode stage for valA
int new_E_valA = [
D_icode in { CALL, JXX } : D_valP; # Use incremented PC
d_srcA == E_dstE : e_valE;    # Forward valE from execute
d_srcA == M_dstM : m_valM;    # Forward valM from memory
d_srcA == M_dstE : M_valE;    # Forward valE from memory
d_srcA == W_dstM : W_valM;    # Forward valM from write back
d_srcA == W_dstE : W_valE;    # Forward valE from write back
```

```
1 : d_rvalA;  # Use value read from register file
];

int new_E_valB = [
d_srcB == E_dstE : e_valE;    # Forward valE from execute
d_srcB == M_dstM : m_valM;    # Forward valM from memory
d_srcB == M_dstE : M_valE;    # Forward valE from memory
d_srcB == W_dstM : W_valM;    # Forward valM from write back
d_srcB == W_dstE : W_valE;    # Forward valE from write back
1 : d_rvalB;  # Use value read from register file
];

################ Execute Stage ######################################

## Select input A to ALU
int aluA = [
E_icode in { RRMOVL, OPL } : E_valA;
E_icode in { IRMOVL, RMMOVL, MRMOVL } : E_valC;
E_icode in { CALL, PUSHL } : -4;
E_icode in { RET, POPL } : 4;
# Other instructions don't need ALU
];

## Select input B to ALU
int aluB = [
E_icode in { RMMOVL, MRMOVL, OPL, CALL,
      PUSHL, RET, POPL } : E_valB;
E_icode in { RRMOVL, IRMOVL } : 0;
# Other instructions don't need ALU
];

## Set the ALU function
int alufun = [
E_icode == OPL : E_ifun;
1 : ALUADD;
];

## Should the condition codes be updated?
bool set_cc = E_icode == OPL;

################ Memory Stage ######################################

## Select memory address
int mem_addr = [
M_icode in { RMMOVL, PUSHL, CALL, MRMOVL } : M_valE;
M_icode in { POPL, RET } : M_valA;
# Other instructions don't need address
];

## Set read control signal
bool mem_read = M_icode in { MRMOVL, POPL, RET };

## Set write control signal
bool mem_write = M_icode in { RMMOVL, PUSHL, CALL };
```