UNIVERSITÉ BORDEAUX 1 Sciences Technologies

Année 2009-2010
Étape : Licence Sciences Technologies (semestre 4)
UE : INF155 (Architecture des ordinateurs)
Corrigé de l'examen du 11 mai 2010

1 Modèles d'architecture (question de cours)

Pipeline Un processeur pipeliné est organisé en étages qui travaillent en parallèle. Les instructions, qui «traversent» le processeur en passant successivement par chaque étage, peuvent donc co-exister avec d'autres instructions se trouvant à des étages différents : le débit du processeur est ainsi augmenté, alors que le temps pendant lequel chaque instruction reste au sein du processeur est légèrement augmenté.

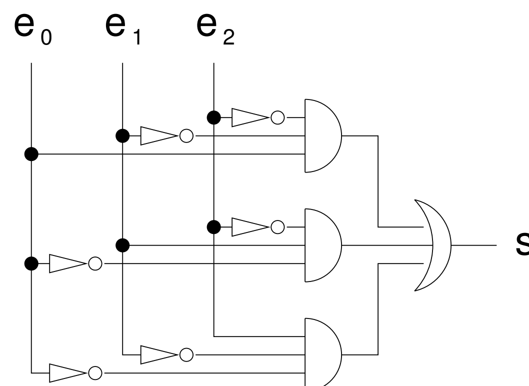
Superscalaire Processeur pipeliné possèdant plusieurs unités de calcul (ALU par exemple), et donc capable d'exécuter plusieurs instructions arithmériques ou logiques simultanément.

HyperThreading Processeur superscalaire qui peut exécuter plusieurs flux d'instructions indépendants (i.e. plusieurs processus). Cela permet d'extrairement davantage de parallélisme entre les instructions que dans seul programme séquentiel.

Multi-cœur Les processeurs multi-cœur sont des puces sur lesquelles sont juxtaposés des «cœurs», qui ne sont rien d'autre que des processeurs...

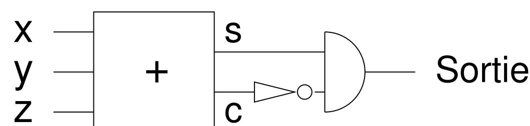
2 Circuits combinatoires

Q1

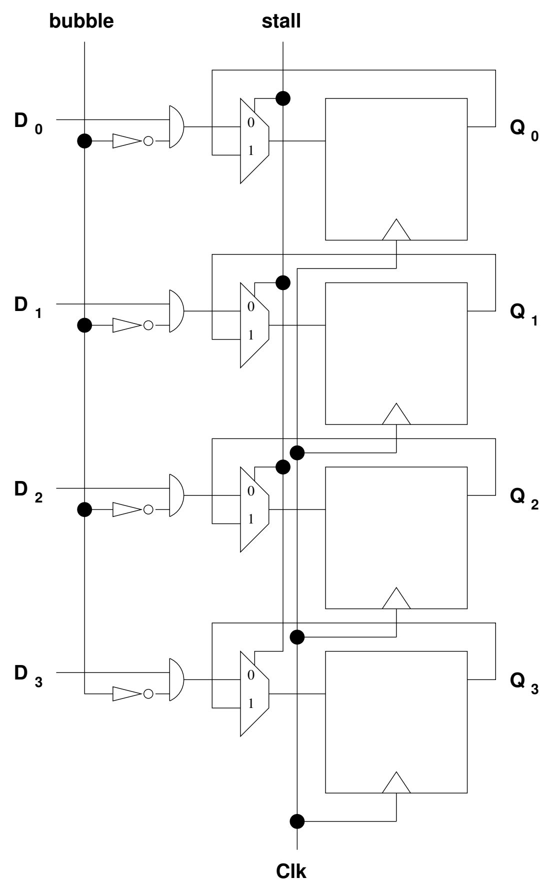


Q2 Le circuit `XOR3` donne un résultat différent en comparaison avec un circuit `XORoXOR` pour les valeurs d'entrée (1, 1, 1) : `XOR3` produit 0 alors que `XORoXOR` produit 1.

Q3



3 Circuits séquentiels



4 Appel de fonction indirect

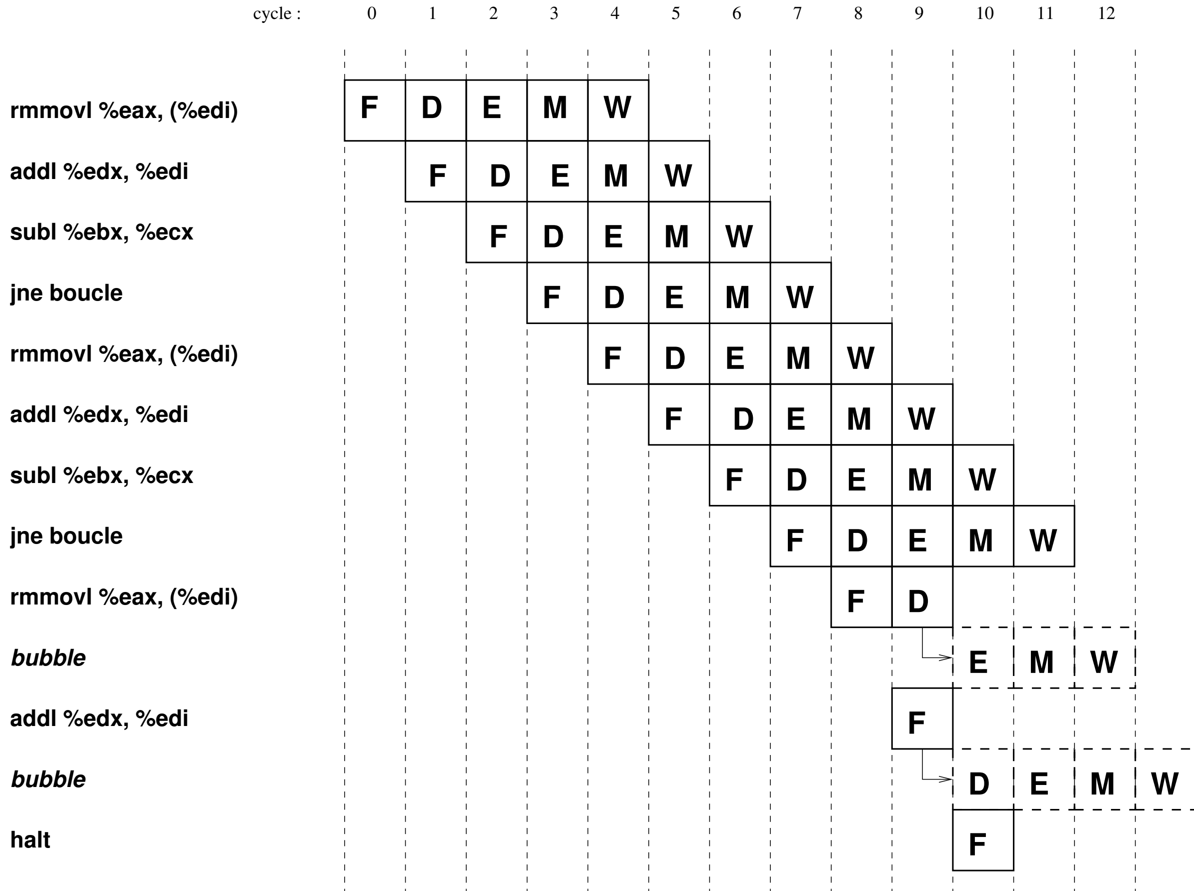
Q1

```
indirect_call:
   popl  %eax   // ret addr
   popl  %edx   // func
   pushl %eax
   pushl %edx
   ret
```

Q2 Il ne faut dépiler que deux paramètres, car `func` a déjà été consommé par une instruction `ret`.

5 Version pipelinée du processeur Y86

Q1 Le programme initialise toutes les cases du tableau «`t`» à zéro. La taille de ce tableau (2) est contenue dans la variable `size`.



L'instruction `halt` est chargée 10 cycles après le chargement de l'instruction `rmmovl`.

Q2 Voir code HCL à la fin de ce corrigé.

Q3 L'intérêt d'utiliser l'instruction `stoswl` est de fusionner deux instructions (`addl` et `rmmovl`) en une seule, grâce au fait que ces instructions n'utilisent pas les mêmes ressources à chaque étage du pipeline. On gagne donc 1 cycle par tour de boucle dans le programme initial, soit 2 cycles au total.

Q4 Voir code HCL à la fin de ce corrigé.

Q5 Il faudrait prévoir un bit supplémentaire `M_ZF` dans le registre intermédiaire mémorisant les valeurs pour l'étage *Memory* dans lequel on récupérerait directement la valeur «`zero flag`» qui sort de l'ALU à l'étage précédent. Ainsi, on pourrait utiliser cette valeur dès que l'instruction `loop` entre à l'étage *Memory* corriger une éventuelle mauvaise prédiction de branchement...

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

## What address should instruction be fetched at
int f_pc = [
        # Mispredicted branch.  Fetch at incremented PC
        M_icode in { JXX, LOOP}  && !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, LOOP } : 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;
        D_icode in { STOSW } : REAX;
        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;
        D_icode in { STOSW } : REDI;
        D_icode in { LOOP } : RECX;
        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;
        D_icode in { STOSW } : REDI;
        D_icode in { LOOP } : RECX;
        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, LOOP } : 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, STOSW } : 4;
        E_icode in { LOOP } : -1;
        # Other instructions don't need ALU
];

## Select input B to ALU
int aluB = [
        E_icode in { RMMOVL, MRMOVL, OPL, CALL,
               PUSHL, RET, POPL, STOSW, LOOP } : 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;
        M_icode in { STOSW } : M_valB;
        # 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, STOSW };
```