CIRC2 : Assembleur, Numération, et Circuits

Licence informatique – semestre 4

Section 1

Multiplication, Division

Multiplication entière (entiers non signés)

`MUL <op1>`

- si `<op1>` est un mot de 8 bits :
  - `<op1>` est multiplié avec le contenu de `al`
  - le résultat est stocké dans `ax`
- si `<op1>` est un mot de 16 bits :
  - `<op1>` est multiplié avec le contenu de `ax`
  - le résultat est stocké dans `dx:ax`
- si `<op1>` est un mot de 32 bits :
  - `<op1>` est multiplié avec le contenu de `eax`
  - le résultat est stocké dans `edx:eax`
- si `<op1>` est un mot de 64 bits :
  - `<op1>` est multiplié avec le contenu de `rax`
  - le résultat est stocké dans `rdx:rax`

Multiplication (entiers non signés) (2) – À vous !

Attention : <op1> ne peut qu'être un registre

Donnez les valeurs contenues dans les registres après les opérations :

- état initial :
  `rax = 0x0000000000001113` et `rdx = 0xFF12000000000011`
- instruction : `MULB %bl  ## avec %bl = $10`
- état final :
  `rax =` et `rdx =`

- état initial :
  `rax = 0xFF34111100111111` et `rdx = 0x000000000000AAAA`
- instruction : `MULW %bx  ## avec %bx = $16`
- état final :
  `rax =` et `rdx =`

Multiplication (entiers non signés) (2) – À vous !

Attention : <op1> ne peut qu'être un registre

Donnez les valeurs contenues dans les registres après les opérations :

- état initial :
  `rax = 0x0000000000001113` et `rdx = 0xFF12000000000011`
- instruction : `MULB %bl  ## avec %bl = $10`
- état final :
  `rax = 0x00000000000000BE` et `rdx = 0xFF12000000000011`

- état initial :
  `rax = 0xFF34111100111111` et `rdx = 0x000000000000AAAA`
- instruction : `MULW %bx  ## avec %bx = $16`
- état final :
  `rax =` et `rdx =`

Multiplication (entiers non signés) (2) – À vous !

Attention : <op1> ne peut qu'être un registre

Donnez les valeurs contenues dans les registres après les opérations :

- état initial :
  `rax = 0x0000000000001113` et `rdx = 0xFF12000000000011`
- instruction : `MULB %bl  ## avec %bl = $10`
- état final :
  `rax = 0x00000000000000BE` et `rdx = 0xFF12000000000011`

- état initial :
  `rax = 0xFF34111100111111` et `rdx = 0x000000000000AAAA`
- instruction : `MULW %bx  ## avec %bx = $16`
- état final :
  `rax = 0xFF34111100111110` et `rdx = 0x0000000000000001`

Multiplication (entiers non signés) (3) – À vous !

- état initial :
  `rax = 0xFF34111100111111` et `rdx = 0x0000000000000000`
- instruction : `MULL %ebx    ## avec %edx = $16`
- état final :
  `rax =` et `rdx =`

- état initial :
  `rax = 0x1011111100111111` et `rdx = 0x0000000000000000`
- instruction : `MULQ %rbx  ## avec %rbx = $16`
- état final :
  `rax =` et `rdx =`

Multiplication (entiers non signés) (3) – À vous !

-
  - état initial :
    `rax = 0xFF34111100111111` et `rdx = 0x0000000000000000`
  - instruction : `MULL %ebx    ## avec %edx = $16`
  - état final :
    `rax = 0xFF34111101111110` et `rdx = 0x0000000000000000`
-
  - état initial :
    `rax = 0x1011111100111111` et `rdx = 0x0000000000000000`
  - instruction : `MULQ %rbx   ## avec %rbx = $16`
  - état final :
    `rax =` et `rdx =`

Multiplication (entiers non signés) (3) – À vous !

- état initial :
  `rax = 0xFF34111100111111` et `rdx = 0x0000000000000000`
- instruction : `MULL %ebx    ## avec %edx = $16`
- état final :
  `rax = 0xFF34111101111110` et `rdx = 0x0000000000000000`

- état initial :
  `rax = 0x1011111100111111` et `rdx = 0x0000000000000000`
- instruction : `MULQ %rbx  ## avec %rbx = $16`
- état final :
  `rax = 0x0111111001111110` et `rdx = 0x0000000000000001`

Division entière (sur entiers non signés)

`DIV <op1>`

- si `<op1>` est un mot de 8 bits :
  - `ax` est divisé par le contenu de `<op1>`
  - le quotient est stocké dans `al`
  - le reste est stocké dans `ah`
- si `<op1>` est un mot de 16 bits :
  - `dx:ax` est divisé par le contenu de `<op1>`
  - le quotient est stocké dans `ax`
  - le reste est stocké dans `dx`
- si `<op1>` est un mot de 32 bits :
  - `edx:eax` est divisé par le contenu de `<op1>`
  - le quotient est stocké dans `eax`
  - le reste est stocké dans `edx`
- si `<op1>` est un mot de 64 bits :
  - `rdx:rax` est divisé par le contenu de `<op1>`
  - le quotient est stocké dans `rax`
  - le reste est stocké dans `rdx`

Division entière (entiers non signés) (2) – À vous !

Attention : <op1> ne peut qu'être un registre

Donnez les valeurs contenues dans les registres après les opérations :

- état initial :
  `rax = 0x0000000000000013` et `rdx = 0xFF12000000000011`
- instruction : `DIVB %bl  ## avec %bl = $10`
- état final :
  `rax =` et `rdx =`

- état initial :
  `rax = 0xFF34111100111111` et `rdx = 0x000000000000000A`
- instruction : `DIVW %bx  ## avec %bx = $16`
- état final :
  `rax =` et `rdx =`

Division entière (entiers non signés) (2) – À vous !

Attention : <op1> ne peut qu'être un registre

Donnez les valeurs contenues dans les registres après les opérations :

- état initial :
  `rax = 0x0000000000000013` et `rdx = 0xFF12000000000011`
- instruction : `DIVB %bl  ## avec %bl = $10`
- état final :
  `rax = 0x0000000000000901` et `rdx = 0xFF12000000000011`

- état initial :
  `rax = 0xFF34111100111111` et `rdx = 0x000000000000000A`
- instruction : `DIVW %bx  ## avec %bx = $16`
- état final :
  `rax =` et `rdx =`

Division entière (entiers non signés) (2) – À vous !

Attention : <op1> ne peut qu'être un registre

Donnez les valeurs contenues dans les registres après les opérations :

- état initial :
  `rax = 0x0000000000000013` et `rdx = 0xFF12000000000011`
- instruction : `DIVB %bl  ## avec %bl = $10`
- état final :
  `rax = 0x0000000000000901` et `rdx = 0xFF12000000000011`

- état initial :
  `rax = 0xFF34111100111111` et `rdx = 0x000000000000000A`
- instruction : `DIVW %bx  ## avec %bx = $16`
- état final :
  `rax = 0xFF3411110011A111` et `rdx = 0x0000000000000001`

Division entière (entiers non signés) (3) – À vous !

- état initial :
  `rax = 0xFF34111100111110` et `rdx = 0x0000000000000010`
- instruction : `DIVL %ebx  ## avec %ebx = $16`
- état final :
  `rax =` et `rdx =`

- état initial :
  `rax = 0x1011111100111111` et `rdx = 0x0000000000000000`
- instruction : `DIVQ  %rbx  ## avec %rbx = $16`
- état final :
  `rax =` et `rdx =`

Division entière (entiers non signés) (3) – À vous !

- état initial :
  `rax = 0xFF34111100111110` et `rdx = 0x0000000000000010`
- instruction : `DIVL %ebx  ## avec %ebx = $16`
- état final :
  `rax = 0xFF34111100011111` et `rdx = 0x0000000000000001`

- état initial :
  `rax = 0x1011111100111111` et `rdx = 0x0000000000000000`
- instruction : `DIVQ  %rbx  ## avec %rbx = $16`
- état final :
  `rax =` et `rdx =`

Division entière (entiers non signés) (3) – À vous !

- état initial :
  `rax = 0xFF34111100111110` et `rdx = 0x0000000000000010`
- instruction : `DIVL %ebx  ## avec %ebx = $16`
- état final :
  `rax = 0xFF34111100011111` et `rdx = 0x0000000000000001`

- état initial :
  `rax = 0x1011111100111111` et `rdx = 0x0000000000000000`
- instruction : `DIVQ  %rbx  ## avec %rbx = $16`
- état final :
  `rax = 0x1011111100011111` et `rdx = 0x0000000000000001`

Section 2

Entiers signés et non signés

Entiers signés et non signés

- Pour le moment, nous n'avons manipulé que des entiers non signés : entiers positifs ou nuls dans $[0; 2^{64} - 1]$
- Les entiers signés permettent de représenter des entiers relatifs. Pour un n bits
  - bit de poids fort pour le signe : 0 entier positif, 1 entier négatif
  - $(n - 1)$ bits pour la valeur absolue :
    - ★ codage comme pour les entiers non signés pour les entiers positifs
    - ★ codage en complément à 2 pour les entiers négatifs

Conserver l'addition

Addition sur des nombres positifs en binaire (notation non signée)

Pour un entier non signé sur n bits, $(2^n - 1) + 1 = 0$ (sur les n bits, avec un problème de débordement).

```
  11111          (1) 1111111        (1) 1111111
  00110110           11111111           11110001
+ 01001101         +  00000001        +  00001111
-----------        -----------        -----------
  10000011       (1) 00000000       (1) 00000000
```

Si on veut conserver le même algorithme d'addition avec les entiers signés, alors `11111111` doit représenter `-1`, et `11110001` doit représenter `-15`...

Notation en complément à deux

Pour calculer la valeur d'un entier signé négatif :

- on inverse la valeur des bits
- on ajoute 1

Entier signé	11111111	11110001
Bit de signe	-1111111	-1110001
Inversion des bits	-0000000	-0001110
Plus 1	-0000001	-0001111
Valeur en décimal	-1	-15

Entiers signés et non signé – À vous !

- Quelle est la plus petite valeur d'un entier signé représenté sur n bits ?

- Quelle est la plus grande valeur d'un entier signé représenté sur n bits ?

Entiers signés et non signé – À vous !

- Quelle est la plus petite valeur d'un entier signé représenté sur *n* bits ?
  exemple sur 8 bits :
  `10000000 = - 1111111 +1 = -10000000` $= -2^7$
  donc $-2^{(n-1)}$
- Quelle est la plus grande valeur d'un entier signé représenté sur *n* bits ?

Entiers signés et non signé – À vous !

- Quelle est la plus petite valeur d'un entier signé représenté sur *n* bits ?
  exemple sur 8 bits :
  `10000000 = - 1111111 +1 = -10000000 =` -2^7
  donc $-2^{(n-1)}$
- Quelle est la plus grande valeur d'un entier signé représenté sur *n* bits ?
  exemple sur 8 bits :
  `01111111 =` $2^7 - 1$
  donc $2^{(n-1)}-1$

Entiers signés et non signé – À vous !

- Quelle est la plus petite valeur d'un entier signé représenté sur *n* bits ?
  exemple sur 8 bits :
  `10000000 = - 1111111 +1 = -10000000 =` -2^7
  donc $-2^{(n-1)}$
- Quelle est la plus grande valeur d'un entier signé représenté sur *n* bits ?
  exemple sur 8 bits :
  `01111111 =` $2^7 - 1$
  donc $2^{(n-1)}-1$

- Les entiers signés sur n bits ont donc une valeur comprise dans l'intervalle $[-2^{(n-1)} ; 2^{(n-1)}-1]$.
- Les entiers positifs (non signés) sur n bits ont une valeur comprise dans l'intervalle $[0 ; 2^n -1]$.

Multiplications et divisions

Les algorithmes de multiplications et de divisions ne peuvent pas être strictement conservés : il faut préciser si on manipule des entiers signés ou non signés.

Multiplication sur des entiers signés : `IMUL <op1>`

Division entière sur des entiers signés : `IDIV <op1>`

Les branchements conditionnels après compraison d'entiers signés

Ils se placent toujours après une instruction `CMP <op1>, <op2>`.
Ils sont toujours de la forme `Jcond <adresse>`.
Ils se lisent *saute à <adresse> si <op2> cond <op1>*

- `JE` : Equal (si les deux opérandes étaient égales) (idem que pour les entiers non signés)
- `JNE` : Not Equal (idem que pour les entiers non signés)
- `JL` et `JNL` : Less (si <op2> est plus petite que <op1>) et Not Less
- `JLE` et `JNLE` : Less or Equal et Not (Less Or Equal)
- `JG` et `JNG` : Greater (si <op2> est plus grande que <op1>) et Not Greater
- `JGE` et `JNGE` : Greater or Equal et Not (Greater Or Equal)

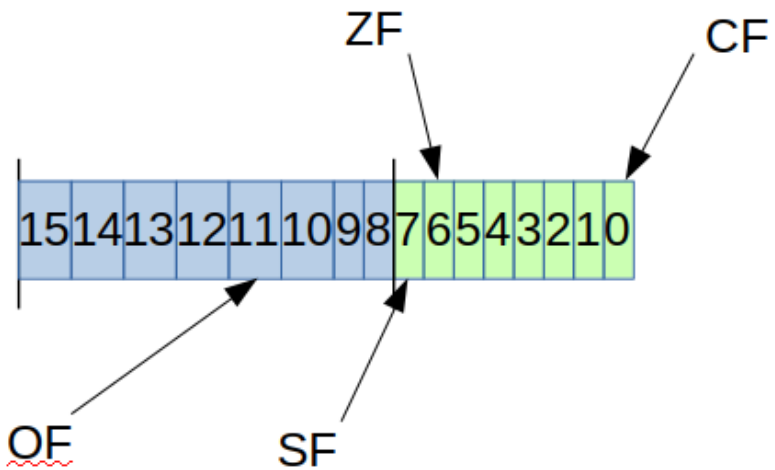
Section 3

La gestion des débordements

Le registre RFLAGS

- registre spécial de 64 bits (le registre EFLAGS est sir 32 bits)
- contient l'état du processeur
- une partie des bits sont des signaux sur le résultat des opérations arithmétiques

Les bits qui nous intéressent sont sur les deux octets de poids faible de `rflags` :



Le registre RFLAGS (2)

- ZF (Zero Flag, bit numéro 6) indique si le résultat est nul (ZF=1) ou non nul (ZF=0).
- SF (Sign Flag, bit numéro 7) indique si le résultat est positif (SF=0) ou négatif (SF=1).
- CF (Carry Flag, bit numéro 0) indique une retenue sur le bit de poids fort pour les entiers non signés (CF=1).
- OF (Overflow Flag, bit numéro 11) indique un débordement (OF=1) sur les entiers signés.

Le registre RFLAGS (2)

- ZF (Zero Flag, bit numéro 6) indique si le résultat est nul (ZF=1) ou non nul (ZF=0).
- SF (Sign Flag, bit numéro 7) indique si le résultat est positif (SF=0) ou négatif (SF=1).
- CF (Carry Flag, bit numéro 0) indique une retenue sur le bit de poids fort pour les entiers non signés (CF=1).
- OF (Overflow Flag, bit numéro 11) indique un débordement (OF=1) sur les entiers signés.

Pour des entiers non signés, un résultat est valide ssi CF vaut 0.
Pour des entiers signés, un résultat est valide ssi OF vaut 0.

Les instructions de branchement conditionnel

Les sauts conditionnels peuvent être exécutés en fonction de l'état de ces flags.

Instruction	Description	Test effectué
`JE`, `JZ`	égalité ou zéro	ZF = 1
`JNE`, `JNZ`	différent ou pas zéro	ZF = 0
`JS`	signe positif	SF = 0
`JNS`	signe négatif	SF = 1
`JC`	retenue (entiers non signés)	CF = 1
`JNC`	pas de retenue (entiers non signés)	CF = 0
`JO`	débordement (entiers signés)	OF = 1
`JNO`	pas de débordement (entiers signés)	OF = 0