TP : 11

Mini-projet avec l'architecture y86(seq)

ARCHITECTURE DES ORDINATEURS

Gabriel MARIE–BRISSON
Clément DELMAS

INF402A52

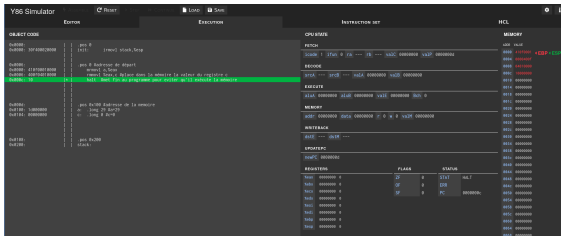
13 avril 2022

Exercice 1 : De la place dans les opcodes y86

Question 1: Dans « Instruction set » nous avons modifié:

rmmovl	4	0	rA,valC
mrmovl	4	1	valC?(rB),rA
mrmovl	4	1	valC,rA

Le code hexadécimale :

Editor	Execution
	

Le code compile bien, mais les instructions rmmovl et mrmovl ne sont plus fonctionnelles.

Question 2: Dans « HCL », exemple de modification:

Y86 par défaut	Modification

`## Does fetched instruction require a register numbers byte?`
`bool need_regids =`
`  icode in { RRMOVL, OPL, IOPL, PUSHL, POPL, IRMOVL, RMMOVL, MRMOVL };`
	`## Does fetched instruction require a register numbers byte?`
	`bool need_regids =`
	`  icode in { RRMOVL, OPL, IOPL, PUSHL, POPL, IRMOVL, RMMOVL };`

`## What register should be used as the M destination?`
`int dstM = [`
`  icode in { MRMOVL, POPL } : rA;`
`  1 : RNONE;  # Don't need register for writing`
`];`
	`## What register should be used as the M destination?`
	`int dstM = [`
	`  icode in { POPL } || (icode == RMMOVL && ifun == 1): rA;`
	`  1 : RNONE;  # Don't need register for writing`
	`];`

`## What register should be used as the A source?`
`int srcA = [`
`  icode in { RRMOVL, OPL, PUSHL } || (icode == RMMOVL) && (ifun == 0) : rA;`
`  icode in { POPL, RET } : RESP;`
`  1 : RNONE;  # Don't need register for reading`
`];`
	`## What register should be used as the A source?`
	`int srcA = [`
	`  icode in { RRMOVL, OPL, PUSHL } || (icode == RMMOVL) && (ifun == 0) : rA;`
	`  icode in { POPL, RET } : RESP;`
	`  1 : RNONE;  # Don't need register for reading`
	`];`

Notre but était de laisser uniquement des RMMOVL. Les deux instructions ont le même icode, mais un ifun différent. Ainsi à tous les endroits on a pu :

-soit factorisé si elles étaient présentes deux fois

-soit les différencier via le ifun si elles étaient présentes une seule fois

Comme vous pouvez le constater dans notre fichier EXO_1, le programme Y86 ce comporte comme voulu. Les valeurs hexadécimale changent au bon endroit. Avec le programme du premier rendu Y86, nous avons comparé les valeurs obtenues entre le simulateur originale et modifié. Elles sont identiques .

Exercice 2 : Factorisation de push/pop/call/ret

Question 1: Nous avons suivi la même logique qu'à l'exercice précédent

Dans « Instruction set » nous avons modifié:

call	8	0	valC
ret	9	0	
pushl	8	1	rA
popl	9	1	rA

Dans « HCL », exemple de modification:

Y86 par précédent	Modification

`## Does fetched instruction require a register numbers byte?`
`bool need_regids =`
`  icode in { RRMOVL, OPL, IOPL, PUSHL, POPL, IRMOVL, RMMOVL };`
	`## Does fetched instruction require a register numbers byte?`
	`bool need_regids =`
	`  icode in { RRMOVL, OPL, IOPL, IRMOVL, RMMOVL } || ((icode == CALL) && (ifun == 1)) || ((icode == RET) && (ifun == 1));`

`## Does fetched instruction require a constant word?`
`bool need_valC =`
`  icode in { IRMOVL, RMMOVL, JXX, CALL, IOPL };`
	`## Does fetched instruction require a constant word?`
	`bool need_valC =`
	`  icode in { IRMOVL, RMMOVL, JXX, IOPL } || ((icode == CALL) && (ifun == 0));`

`## What register should be used as the A source?`
`int srcA = [`
`  icode in { RRMOVL, OPL, PUSHL } || (icode == RMMOVL) && (ifun == 0) : rA;`
`  icode in { POPL, RET } : RESP;`
`  1 : RNONE;  # Don't need register for reading`
`];`
	`## What register should be used as the A source?`
	`int srcA = [`
	`  icode in { RRMOVL, OPL} || ((icode == RMMOVL) && (ifun == 0))  || ((icode == CALL) && (ifun == 1)) : rA;`
	`  icode in { RET } : RESP;`
	`  1 : RNONE;  # Don't need register for reading`
	`};`

`## What register should be used as the E destination?`
`int dstE = [`
`  icode in { RRMOVL, IRMOVL, OPL, IOPL } : rB;`
`  icode in { PUSHL, POPL, CALL, RET } : RESP;`
`  1 : RNONE;  # Don't need register for writing`
`];`
	`## What register should be used as the E destination?`
	`int dstE = [`
	`  icode in { RRMOVL, IRMOVL, OPL, IOPL } : rB;`
	`  icode in { RET, CALL }: RESP;`
	`  1 : RNONE;  # Don't need register for writing`
	`];`

Code de test n°1:

Nous avons testé les modifications avec deux codes différents. Celui étant le plus simple, il est plus facile de regarder l'état de la pile. Par la suite nous avons testé avec le second rendu en Y86. Les valeurs obtenues sont les mêmes que sans les modifications.

Question 2 bonus:

Dans « Instruction set » nous avons modifié:

call	8	0	valC
ret	8	1	
pushl	8	2	rA
popl	8	3	rA

Pour facilité les modifications dans le HCL, nous avons choisies cette implémentation.

Dans « HCL », exemple de modification:

Y86 par précédent	Modification

`## Does fetched instruction require a register numbers byte?`
`bool need_regids =`
`  icode in { RRMOVL, OPL, IOPL, IRMOVL, RMMOVL } || ((icode == CALL) && (ifun == 1)) || ((icode == RET) && (ifun == 1));`
	`## Does fetched instruction require a register numbers byte?`
	`bool need_regids =`
	`  icode in { RRMOVL, OPL, IOPL, IRMOVL, RMMOVL } || ((icode == CALL) && (ifun == 2)) || ((icode == CALL) && (ifun == 3));`

`## Does fetched instruction require a constant word?`
`bool need_valC =`
`  icode in { IRMOVL, RMMOVL, JXX, IOPL } || ((icode == CALL) && (ifun == 0));`
	`## Does fetched instruction require a constant word?`
	`bool need_valC =`
	`  icode in { IRMOVL, RMMOVL, JXX, IOPL } || ((icode == CALL) && (ifun == 0));`

`## What register should be used as the A source?`
`int srcA = [`
`  icode in { RMMOVL, OPL} || ((icode == RMMOVL) && (ifun == 0))  || ((icode == CALL) && (ifun == 1)) : rA;`
`  icode in { RET } : RESP;`
`  1 : RNONE;  # Don't need register for reading`
`];`
	`## What register should be used as the A source?`
	`int srcA = [`
	`  icode in { RMMOVL, OPL} || ((icode == RMMOVL) && (ifun == 0))  || ((icode == CALL) && (ifun == 2)) : rA;`
	`  ((icode == CALL) && (ifun == 1)) || ((icode == CALL) && (ifun == 3)) : RESP;`
	`  1 : RNONE;  # Don't need register for reading`
	`];`

`## What register should be used as the E destination?`
`int dstE = [`
`  icode in { RRMOVL, IRMOVL, OPL, IOPL } : rB;`
`  icode in { RET, CALL }: RESP;`
`  1 : RNONE;  # Don't need register for writing`
`];`
	`## What register should be used as the E destination?`
	`int dstE = [`
	`  icode in { RRMOVL, IRMOVL, OPL, IOPL } : rB;`
	`  icode in { CALL }: RESP;`
	`  1 : RNONE;  # Don't need register for writing`
	`];`

Nous avons passé les mêmes programmes, on a retrouvé les bons résultats.

Exercice 3 : Ajout de l'instruction loop

Question 1: Nous avons rajouté l'instruction loop, choisie le premier icode disponible puis mit le ifun a 0 pour garder la logique précédente.

loop	9	0	valC

Question 2:

Dans « HCL », exemple de modification:

Ensemble des modifications

```
## Is instruction valid?
bool instr_valid =
    icode in { NOP, HALT, RRMOVL, IRMOVL, RMMOVL,
               OPL, IOPL, JXX, CALL,LOOP} ;

## Decode stage ######################################
```

```
## Does fetched instruction require a constant word?
bool need_valC =
    icode in { IRMOVL, RMMOVL, JXX, IOPL ,LOOP} || ((icode == CALL) && (ifun == 0));
```

```
## What register should be used as the B source?
int srcB = [
    icode in { OPL, IOPL, RMMOVL} : rB;
    icode in { CALL } : RESP;
    (icode == LOOP) : RECX;
    1 : RNONE;  # Don't need register for reading
];
```

```
## Select input A to ALU
int aluA = [
    icode in { RRMOVL, OPL } : valA;
    icode in { IRMOVL, RMMOVL, IOPL } : valC;
    ((icode == CALL) && (ifun == 0)) || ((icode == CALL) && (ifun == 2)) : -4;
    ((icode == CALL) && (ifun == 1)) || ((icode == CALL) && (ifun == 3)) : 4;
    icode in { LOOP } : -1;
    # Other instructions don't need ALU
];
```

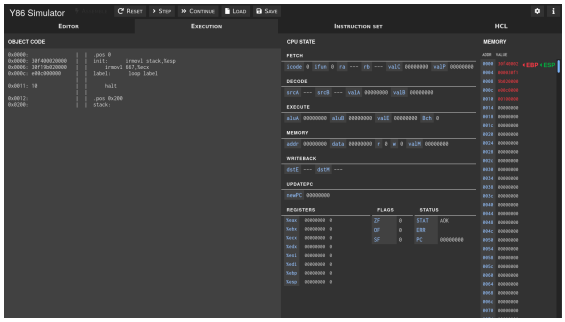
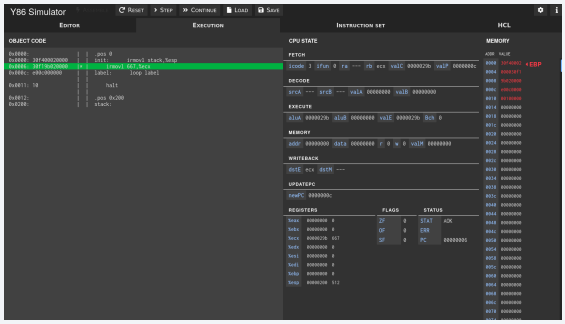
```
## Select input B to ALU
int aluB = [
    icode in { RMMOVL, OPL, IOPL, CALL, LOOP } : valB;
    icode in { RRMOVL, IRMOVL } : 0;
    # Other instructions don't need ALU
];
```

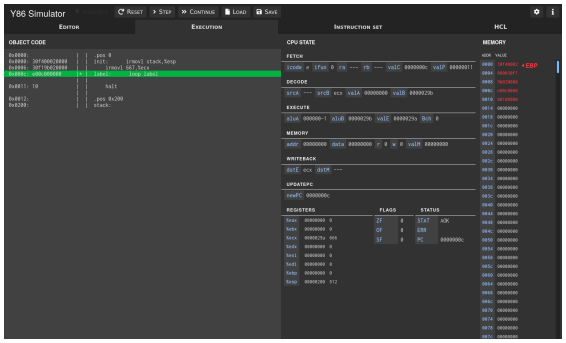
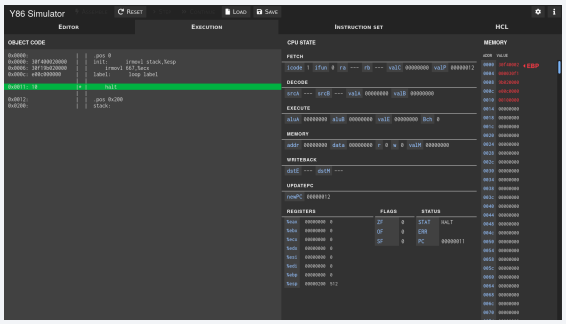
Ensemble des modifications

```
## What register should be used as the E destination?
int dstE = [
     icode in { RRMOVL, IRMOVL, OPL, IOPL } : rB;
     icode in { CALL }: RESP;
     (icode == LOOP) : RECX;
     1 : RNONE;  # Don't need register for writing
];
```

```
## Compute address of next instruction to be fetched
int new_pc = [
     # Call: Use immediate value
     (icode == CALL && ifun == 0) : valC;
     # Taken branch:  Use immediate value
     icode == JXX && Bch : valC;
     # Completion of RET instruction: Use value retrieved from stack
     (icode == CALL && ifun == 1)  : valM;
     ((icode == LOOP) && (valE > 0)) : valC;
     # Default: Use incremented PC
     1 : valP;
];
```

Etape de l'exécution du programme de test de loop

	Etat initiale du similiateur
	Ajout du init 667 dans le registre du %ecx

	Boucle loop décrémentant un par un %ecx
	Quand %ecx égale 0 sortie de la boucle et arrêt du programme. Nous remarquons qu'il n'y a pas eu de modification des flags.

Exercice 4 : Ajout des instructions loope/loopne

Question 1:

Dans « Instruction set » nous avons modifié:

loop	9	0	valC
loope	9	3	valC
loopne	9	4	valC

Nous avons mis le premier icode disponible. Le 3 correspond à l'icode de je. Le 4 correspond à l'icode de jne. Ceci permet au HCL de différencier les instructions.

Question 2:

On ajoute LOOP dans is_bch pour que bch soit a un des qu'il y a une loop et il faut donc que dans new_pc on vérifie l'état de Bch.

Ensemble des modifications

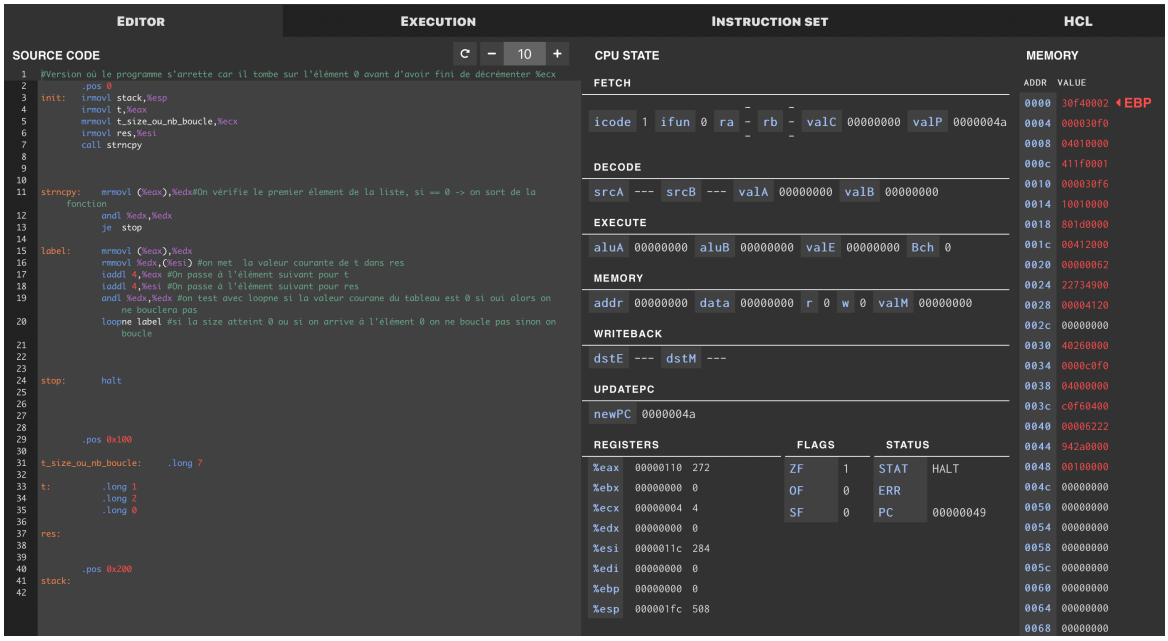
```
bool is_bch = icode in { JXX, LOOP};
```

```
## Compute address of next instruction to be fetched
int new_pc = [
    # Call: Use immediate value
    (icode == CALL && ifun == 0) : valC;
    # Taken branch:  Use immediate value
    icode == JXX && Bch : valC;
    # Completion of RET instruction: Use value retrieved from stack
    (icode == CALL && ifun == 1)  : valM;
    icode == LOOP && Bch && valE > 0: valC;
    # Default: Use incremented PC
    1 : valP;
];
```

Question 3:

On a crée deux programme pour illustrer le fonctionnement de loopne. Le premier s'arrête parce que l'élément 0 à été trouver dans le tableau t (avant que %ecx vaille 0) et le second s'arrête car %ecx est décrémenter à 0 avant de trouver l'élément 0 dans le tableau t.

Programme n°1:



Programme n°2: