

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
rrmmovl	4	1	valC?(rB), rA
rrmmovl	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
<pre>## Does fetched instruction require a register numbers byte? bool need_regids = icode in { RRMovl, OPl, IOPl, PUSHl, POPl, IRMovl, RMMovl, MRMOVL };</pre>	<pre>## Does fetched instruction require a register numbers byte? bool need_regids = icode in { RRMovl, OPl, IOPl, PUSHl, POPl, IRMovl, RMMovl };</pre>
<pre>## What register should be used as the M destination? int dstM = [icode in { MRMOVL, POPl } : rA; 1 : RNONE; # Don't need register for writing];</pre>	<pre>## 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];</pre>
<pre>## 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];</pre>	<pre>## 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];</pre>

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
<pre>## Does fetched instruction require a register numbers byte? bool need_regids = icode in { RRMovl, OPl, IOPl, PUSHl, POPl, IRMovl, RMMovl };</pre>	<pre>## 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));</pre>
<pre>## Does fetched instruction require a constant word? bool need_valC = icode in { IRMovl, RMMovl, JXX, CALL, IOPl };</pre>	<pre>## Does fetched instruction require a constant word? bool need_valC = icode in { IRMovl, RMMovl, JXX, IOPl } ((icode == CALL) && (ifun == 0));</pre>
<pre>## 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];</pre>	<pre>## 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];</pre>
<pre>## 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];</pre>	<pre>## 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];</pre>

Code de test n°1:

EDITOR	EXECUTION
<pre>SOURCE CODE 1 .pos 0 2 init: irmovl stack,%esp 3 irmovl 5,%eax 4 pushl %eax 5 popl %ebx 6 call test 7 halt 8 9 test: irmovl 1,%ecx 10 11 .pos 0x200 12 stack: 13</pre>	

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

Dans « HCL », exemple de modification:

Y86 par précédent	Modification
<pre>## Does fetched instruction require a register numbers byte? bool need_regids = icode in { RMOVL, OPL, IOPL, IRMOVL, RMOVL } ((icode == CALL) && (ifun == 1)) ((icode == RET) && (ifun == 1));</pre>	<pre>## Does fetched instruction require a register numbers byte? bool need_regids = icode in { RMOVL, OPL, IOPL, IRMOVL, RMOVL } ((icode == CALL) && (ifun == 2)) ((icode == CALL) && (ifun == 3));</pre>
<pre>## Does fetched instruction require a constant word? bool need_valC = icode in { IRMOVL, RMMOVL, JXX, IOPL } ((icode == CALL) && (ifun == 0));</pre>	<pre>## Does fetched instruction require a constant word? bool need_valC = icode in { IRMOVL, RMMOVL, JXX, IOPL } ((icode == CALL) && (ifun == 0));</pre>
<pre>## What register should be used as the A source? int srcA = 1; icode in { RMOVL, OPL } ((icode == RMMOVL) && (ifun == 0)) ((icode == CALL) && (ifun == 1)) : rA; icode in { RET } : RESP; 1 : RNONE; # Don't need register for reading</pre>	<pre>## What register should be used as the A source? int srcA = 1; icode in { RMOVL, OPL } ((icode == RMMOVL) && (ifun == 0)) ((icode == CALL) && (ifun == 2)) : rA; ((icode == CALL) && (ifun == 3)) ((icode == CALL) && (ifun == 3)) : RESP; 1 : RNONE; # Don't need register for reading</pre>
<pre>## What register should be used as the E destination? int dstE = 1; icode in { RMMOVL, IRMOVL, OPL, IOPL } : rB; icode in { RET, CALL } : RESP; 1 : RNONE; # Don't need register for writing</pre>	<pre>## What register should be used as the E destination? int dstE = 1; icode in { RMMOVL, IRMOVL, OPL, IOPL } : rB; icode in { CALL } : RESP; 1 : RNONE; # Don't need register for writing</pre>

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
];
```

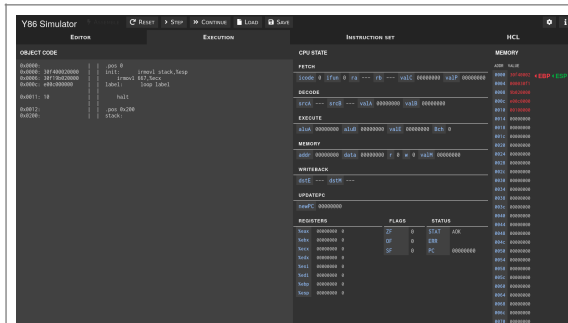
```
## Select input B to ALU
int aluB = [
    icode in { RMMOVL, OPL, IOPL, CALL, LOOP } : valB;
    icode in { RRMovL, IRMOVL } : 0;
    # Other instructions don't need ALU
];
```

Ensemble des modifications

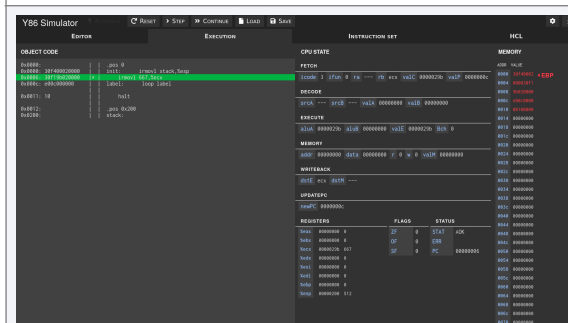
```
## What register should be used as the E destination?
int dstE = [
    icode in { RRMOVL, IRMOVL, OPL, IOPL } : rB;
    icode in { CALL } : RESP;
    (icode == LOOP) : RECX;
    1 : RNONE; # Don't need register for writing
];
```

```
## 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;
];
```

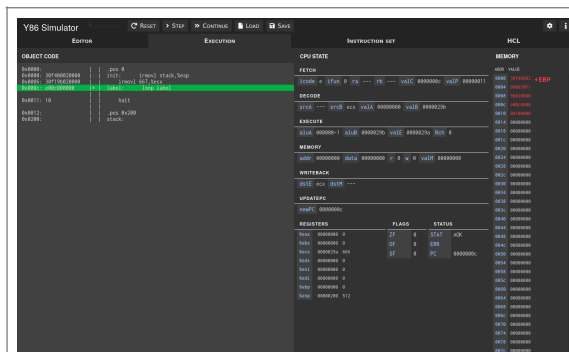
Etape de l'exécution du programme de test de loop



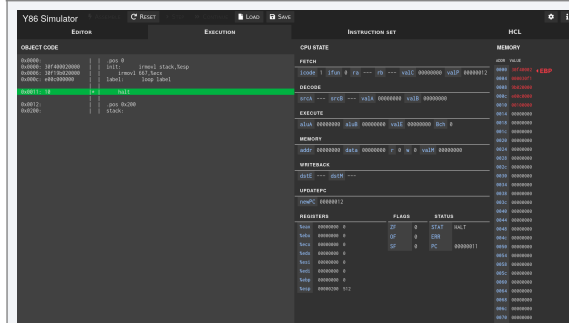
Etat initiale du simulateur



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 loop/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	
<pre>bool is_bch = icode in { JXX, LOOP};</pre>	<pre>## 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;];</pre>

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:

EDITOR	EXECUTION	INSTRUCTION SET	HCL
SOURCE CODE	CPU STATE	MEMORY	
1 #Version ou le programme s'arrete car il tombe sur l'élément 0 avant d'avoir fini de décrémenter %ecx	FETCH	icode 1 ifun 0 ra - rb - valC 00000000 valP 0000004a	ADDR VALUE 0000 30f40002 ◀EBP
2 .pos 0	DECODE	srcA --- srcB --- valA 00000000 valB 00000000	0004 000030f0
3 init: immovl stack,%esp	EXECUTE	aluA 00000000 aluB 00000000 valE 00000000 Bch 0	0008 04010000
4 immovl t,%ecx	MEMORY	addr 00000000 data 00000000 r 0 w 0 valM 00000000	000c 411f0001
5 mmovl t_size_ou_nb_boucle,%ecx	WRITEBACK	dstE --- dstM ---	0010 000030f6
6 immovl res,%esi	UPDATEPC	newPC 0000004a	0014 10010000
7 call strncpy	REGISTERS		0018 801d0000
8			001c 00412000
9			0020 00000062
10			0024 22734000
11 strncpy: mmovl (%eax),%edx#on verifie le premier élément de la liste, si == 0 -> on sort de la fonction			0028 00004120
12 andl %edx,%edx			002c 00000000
13 je stop			0030 40260000
14			0034 0000c0f0
15 label: mmovl (%eax),%ecx			0038 04000000
16 mmovl %ecx,%esi#on met la valeur courante de t dans res			003c c0f60400
17 addl 4,%ecx#on passe à l'élément suivant pour t			0040 00006222
18 addl 4,%esi#on passe à l'élément suivant pour res			0044 942a0000
19 andl %edx,%edx#on test avec loopne si la valeur courante du tableau est 0 si oui alors on ne bouclera pas			0048 00100000
20 loopne label#si la size atteint 0 ou si on arrive à l'élément 0 on ne boucle pas sinon on boucle			004c 00000000
21			0050 00000000
22			0054 00000000
23			0058 00000000
24 stop: halt			005c 00000000
25			0060 00000000
26			0064 00000000
27			0068 00000000
28 .pos 0x100			
29			
30 t_size_ou_nb_boucle: .long 7			
31			
32 t: .long 1			
33 .long 2			
34 .long 0			
35			
36 res: .pos 0x200			
37			
38			
39			
40			
41			
42			
43			
44			
45			

Programme n°2:

EDITOR	EXECUTION	INSTRUCTION SET	HCL
SOURCE CODE	CPU STATE	MEMORY	
1 #Version ou le programme s'arrete car il fini de décrémenter %ecx avant de tomber sur l'élément 0	FETCH	icode 1 ifun 0 ra - rb - valC 00000000 valP 0000004a	ADDR VALUE 0000 30f40002 ◀EBP
2 .pos 0	DECODE	srcA --- srcB --- valA 00000000 valB 00000000	0004 000030f0
3 init: immovl stack,%esp	EXECUTE	aluA 00000000 aluB 00000000 valE 00000000 Bch 0	0008 04010000
4 immovl t,%ecx	MEMORY	addr 00000000 data 00000000 r 0 w 0 valM 00000000	000c 411f0001
5 mmovl t_size_ou_nb_boucle,%ecx	WRITEBACK	dstE --- dstM ---	0010 000030f6
6 immovl res,%esi	UPDATEPC	newPC 0000004a	0014 1c010000
7 call strncpy	REGISTERS		0018 801d0000
8			001c 00412000
9			0020 00000062
10			0024 22734000
11 strncpy: mmovl (%eax),%edx#on verifie le premier élément de la liste, si == 0 -> on sort de la fonction			0028 00004120
12 andl %edx,%edx			002c 00000000
13 je stop			0030 40260000
14			0034 0000c0f0
15 label: mmovl (%eax),%ecx			0038 04000000
16 mmovl %ecx,%esi#on met la valeur courante de t dans res			003c c0f60400
17 addl 4,%ecx#on passe à l'élément suivant pour t			0040 00006222
18 addl 4,%esi#on passe à l'élément suivant pour res			0044 942a0000
19 andl %edx,%edx#on test avec loopne si la valeur courante du tableau est 0 si oui alors on ne bouclera pas			0048 00100000
20 loopne label#si la size atteint 0 ou si on arrive à l'élément 0 on ne boucle pas sinon on boucle			004c 00000000
21			0050 00000000
22			0054 00000000
23			0058 00000000
24 stop: halt			005c 00000000
25			0060 00000000
26			0064 00000000
27			0068 00000000
28 .pos 0x100			
29			
30 t_size_ou_nb_boucle: .long 2			
31			
32 t: .long 1			
33 .long 2			
34 .long 43			
35 .long 21			
36 .long 23			
37 .long 0			
38			
39			
40			
41			
42			
43			
44			
45			