

Executable semantics of Arm's Architecture Specification Language and application to CAS

Hadrien Renaud

University College London

JFLA 2024

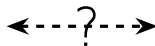
Objectifs

CAS X_s , X_t , $[X_n]$

Reference
Manual

herd7

```
let address = X[n];
let compare_value = X[s];
let new_value = X[t];
let data = Mem[address];
if data == compare_value then
  Mem[address] = new_value;
end
X[s] = data;
```



Objectifs

CAS X_s , X_t , $[X_n]$

Reference
Manual

herd7

```
let address = X[n];
let compare_value = X[s];
let new_value = X[t];
let data = Mem[address];
if data == compare_value then
  Mem[address] = new_value;
end
X[s] = data;
```



Compare-And-Swap CAS X_s , X_t , $[X_n]$ - sémantique informelle

Approximativement :

- Lire l'adresse x depuis le registre X_n .
- Lire la valeur à comparer v_c depuis le registre X_s .
- Lire la nouvelle valeur v_n depuis le registre X_t .
- Lire la valeur v_m dans la case mémoire est donnée par l'adresse x .
- Si $v_m = v_c$, écrire la valeur v_n en mémoire à l'adresse x .
- Écrire la valeur v_m dans le registre X_s .

Compare-And-Swap CAS X_s , X_t , $[X_n]$ - sémantique informelle

Approximativement :

- Lire l'adresse x depuis le registre X_n .
- Lire la valeur à comparer v_c depuis le registre X_s .
- Lire la nouvelle valeur v_n depuis le registre X_t .
- Lire la valeur v_m dans la case mémoire est donnée par l'adresse x .
- Si $v_m = v_c$, écrire la valeur v_n en mémoire à l'adresse x .
- Écrire la valeur v_m dans le registre X_s .

2 cas :

“ok” si la condition est vraie, et que l'écriture mémoire se produit ;
“no” sinon.

Compare-And-Swap CAS X_s , X_t , $[X_n]$ en ASL

Code ASL simplifié :

```
let address = X[n];
let compare_value = X[s];
let new_value = X[t];
let data = Mem[address];
if data == compare_value then
  Mem[address] = new_value;
end
X[s] = data;
```

Compare-And-Swap CAS X_s , X_t , $[X_n]$ en ASL

Code ASL simplifié :

```
let address = X[n];  
let compare_value = X[s];  
let new_value = X[t];  
let data = Mem[address];  
if data == compare_value then  
  Mem[address] = new_value;  
end  
X[s] = data;
```

Lire x depuis le registre X_n .
Lire v_c depuis le registre X_s .
Lire v_n depuis le registre X_t .
Lire v_m en mémoire à l'adresse x .
Si $v_m = v_c, \dots$
écrire v_n en mémoire à l'adresse x .
Écrire v_m dans le registre X_s .

Compare-And-Swap CAS X_s , X_t , $[X_n]$

Graphe généré par herd7 :

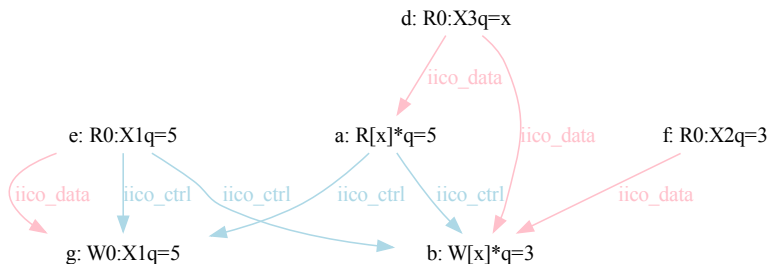


Figure – Graphe généré par herd7 pour l’instruction CAS X_1 , X_2 , $[X_3]$ dans le cas ok

Objectifs

```
let address = X[n];
let compare_value = X[s];
let new_value = X[t];
let data = Mem[address];
if data == compare_value then
  Mem[address] = new_value;
end
X[s] = data;
```



Sémantique des instructions AArch64

Dépendances d'adresse ou de donnée

Ordonne une lecture mémoire et un autre accès mémoire.

Inclus des *Dépendances intra-instruction de donnée*.

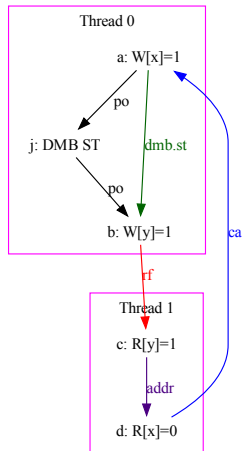
Exemple :

AArch64 MP+dmb+addr

```
{
  0: X0 = x ; 1: X0 = x ;
  0: X1 = y ; 1: X1 = y ;
}
```

```
P0 | P1 ;
MOV W2, #1 | LDR W2, [X1] ;
STR W2, [X0] | EOR W4, W2, W2 ;
DMB ST | ORR X0, X0, X4 ;
STR W2, [X1] | LDR W3, [X0] ;
```

```
exists (1: X2 = 1 /\ 1: X3 = 0)
```



Dépendances d'adresse ou de donnée

Ordonne une lecture mémoire et un autre accès mémoire.

Inclus des *Dépendances intra-instruction de donnée*.

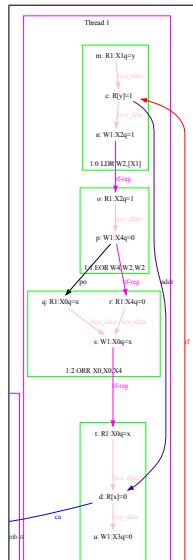
Exemple :

AArch64 MP+dmb+addr

```
{  
  0: X0 = x ; 1: X0 = x ;  
  0: X1 = y ; 1: X1 = y ;  
}
```

```
P0 | P1 ;  
MOV W2, #1 | LDR W2, [X1] ;  
STR W2, [X0] | EOR W4, W2, W2 ;  
DMB ST | ORR X0, X0, X4 ;  
STR W2, [X1] | LDR W3, [X0] ;
```

```
exists (1: X2 = 1 /\ 1: X3 = 0)
```



Objectifs

CAS X_s , X_t , $[X_n]$

Reference
Manual

herd7

```
let address = X[n];
let compare_value = X[s];
let new_value = X[t];
let data = Mem[address];
if data == compare_value then
  Mem[address] = new_value;
end
X[s] = data;
```



Dépendances intra-instruction de données

Intuition

Pour a, b deux évènements :

$a \xrightarrow{iico_data} b \iff b \text{ utilise des données définies dans } a$

Exemple : LDR Xd, [Xn]

```
let address = X[n];  
let data = Mem[address];  
X[d] = data;
```

c: R0:X0q=x

\downarrow
iico_data

a: R[x]q=1

\downarrow
iico_data

d: W0:X2q=1

Dépendances intra-instruction de contrôle

Une autre sorte de dépendances intra-instructions !

Intuition

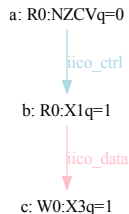
Pour a, b deux évènements :

$$a \xrightarrow{iico_ctrl} b \iff b \text{ est dans un } if \text{ dont la condition utilise } a$$

Exemple : CSEL X_d , X_n , X_m , $cond$

```
var result: bits(64);
if ConditionHolds(cond) then
  result = X[n];
else
  result = X[m];
end

X[d] = result;
```



Objectifs

CAS $X_s, X_t, [X_n]$

Reference
Manual

herd7

```
let address = X[n];  
let compare_value = X[s];  
let new_value = X[t];  
let data = Mem[address];  
if data == compare_value then  
  Mem[address] = new_value;  
end  
X[s] = data;
```

ASLRef

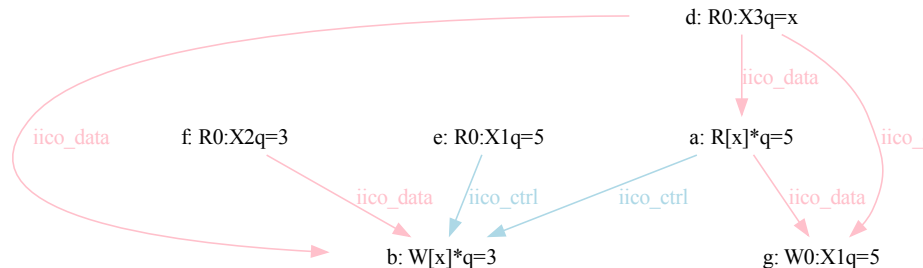


Application à CAS

Code ASL simplifié :

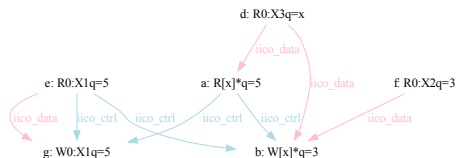
```
let address = X[n];
let compare_value = X[s];
let new_value = X[t];
let data = Mem[address];
if data == compare_value then
  Mem[address] = new_value;
end
X[s] = data;
```

Graphe généré :



Oh oh, il y a une divergence

Graphe généré par herd7 :

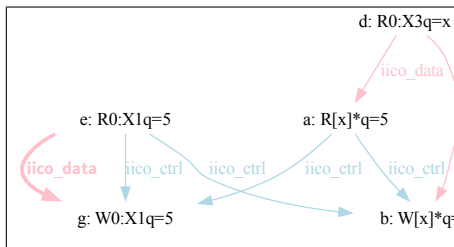


Graphe généré depuis le code ASL :

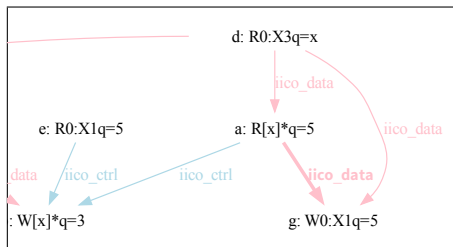


Oh oh, il y a une divergence

Graphe généré par herd7 :



Graphe généré depuis le code ASL :



Compare-And-Swap CAS Xs , Xt , $[Xn]$ - sémantique informelle

Approximativement :

- Lire l'adresse x depuis le registre Xn .
- Lire la valeur à comparer v_c depuis le registre Xs .
- Lire la nouvelle valeur v_n depuis le registre Xt .
- Lire la valeur v_m dans la case mémoire est donnée par l'adresse x .
- Si $v_m = v_c$, écrire la valeur v_n en mémoire à l'adresse x .
- Écrire la valeur v_m dans le registre Xs .

On est ici dans le cas où $v_m = v_c$!

Est-ce que c'est grave ?

On écrit la même valeur ...

Est-ce que c'est grave ?

On écrit la même valeur ...

Ces programmes ont des comportements différents en fonction de quelle flèche est là :

```
AArch64 MP+rel+CAS-ok-RsRs-addr
{
int z=1;
0:X0=x; 0:X2=y;
1:X0=x; 1:X2=y; 1:X4=z;
}
```

```
P0          | P1          ;
MOV W1,#1   | LDR W1,[X2] ;
            | MOV W9,W1   ;
STR W1,[X0] | CAS W1,W3,[X4] ;
MOV W3,#1   | AND W5,W1,#2 ;
STLR W3,[X2]| LDR W7,[X0,W5,SXTW] ;

exists 1:X9=1 /\ 1:X7=0
```

```
AArch64 MP+rel+CAS-ok-MRs-addr
{
0:X0=x; 0:X1=y;
1:X0=x; 1:X1=y;
}
```

```
P0          | P1          ;
MOV W2, #1   | MOV W2, #1   ;
STR W2, [X1] | CAS W2, WZR, [X0] ;
STLR W2, [X0]| EOR W3, W2, W2 ;
            | LDR W4, [X1,W3,SXTW] ;

exists 1:X2=1 /\ 1:X4=0
```

À qui la faute ?

ASL ou herd7 ?

À qui la faute ?

ASL ou herd7 ?

Les deux sont valides : choix non déterministe entre les deux graphes !

Rafiner l'intention des architectes pour la sémantique des instructions.

Conclusion

- Patché dans herd7, discussion en cours dans Arm pour ASL.
- Code de l'interpréteur dans
<https://github.com/herd/herdtools7/blob/master/asllib>.
- Documents de référence ASL
<https://developer.arm.com/documentation/DDI0621/00alp0>.
- Beaucoup de questions de type-checking, existent aussi au même endroit
<https://developer.arm.com/documentation/DDI0622/00alp0>.
- Projet plus vaste mené à Arm avec :

Jade Alglave
j.alglave@ucl.ac.uk
memory-model@arm.com

Luc Maranget
Luc.Maranget@inria.fr

et moi
hadrien.renaud.22@ucl.ac.uk
hadrien.renaud2@arm.com