情報学群実験第 2 レポート

ARM アセンブリ言語の即値代入制限に関する実験

1250373 溝口洸熙[*1]

2022 年 12 月 5 日

[*1] 高知工科大学 情報学群 2 年 清水研究室

第 1 章

即値代入の制限における規則性

1.1　実験の目的

arm Assembler の mov 命令[*1]は，i386 Assembler で扱った mov 命令と対応する．しかし，arm Assembler の mov 命令において，レジスタに代入できる即値には制限がある．本実験の目的は，その制限について実験に基づき示すことである．

1.2　実験の方法

コンピュータ（src 1）を利用して，mov.s（src 3）ファイルをアセンブルする．

src 1　利用コンピュータ及び実行環境

```
$ uname -a
Linux KUT20VLIN-322 5.4.0-70-generic #78~18.04.1-Ubuntu SMP Sat Mar 20 14:10:07 UTC
2021 x86_64 x86_64 x86_64 GNU/Linux
$ arm-none-eabi-as --version
GNU assembler (2,27-9ubuntu1+9) 2.27
$ bash --version
GNU bash, バージョン 4.4.20(1)-release (x86_64-pc-linux-gnu)
```

src 2　アセンブル

```
1 $ arm-none-eabi-as mov.s -o mov.o
2 $ arm-none-eabi-ld mov.o -o mov
3 $ ./mov ; echo $?
4 3
```

src 2: 4 行目は，src 3: 5 行目のテスト値を出力する．

アセンブル（src 2）の際に src 4 の出力が得られた場合は，src 3: 5 行目に問題があり即値の代入に失敗している．

src 3　mov.s

```
1     .section    .text
2     .global     _start
3 _start:
4     mov r7, #1
5     mov r0, #3 @ test number
6     swi #0
```

src 4　Error 出力

```
mov.s: Assembler messages:
mov.s:5: Error: invalid constant (4d1)
   after fixup
```

[*1] レジスタに即値，またはレジスタの値を代入する命令．

様々なテスト値を簡単に試すためにスクリプトファイルによる自動実行プログラム（src 7）を作成し，実行した[*2]（src 6）．テストする即値は，$1 \leq N \leq 2^{25}$ である．また，2^{21} 以降は処理に時間がかかりすぎたため，24 ビット目に着目して手作業でテストした．実行結果には，アセンブルできた即値とその即値の前にアセンブルできた即値との差（前項との差）が出力される．

src 5　mov.s 変更後

```
1      .equ     N,   %d
2      .section     .text
3      .global      _start
4  _start:
5      mov r7, #1
6      mov r0, #N @ test number
7      swi #0
```

src 6　実行

```
$ bash test.sh
```

src 7　test.sh

```
1  #!/bin/bash
2  W=0
3  for i in 'seq 0 33554432'
4  do
5  sed -e "s/%d/$i/g" mov.s > test.s
6  arm-none-eabi-as test.s -o test.o > /dev/null 2>&1
7  if [ $? -eq 0]; then
8      echo "$i, OK, $(( $i - $W ))"
9      V=$i
10     W-$i
11 fi
12 done
```

1.3　実験結果

実験結果を Tbl 1，Tbl 2 に示す．

Tbl 1　規則的な実験結果

入力数値	アセンブルの可否	前項との差
0	OK	-
1	OK	1
2	OK	1
⋮	⋮	⋮
256	OK	1
260	OK	4
⋮	⋮	⋮
1024	OK	4
1040	OK	16
⋮	⋮	⋮
4096	OK	16
4160	OK	64
⋮	⋮	⋮
16384	OK	64
16640	OK	256
⋮	⋮	⋮
66536	OK	256
65569	OK	1024
⋮	⋮	⋮
2^{24}	OK	4^9

Tbl 2　例外的な実験結果

入力数値	アセンブルの可否	前項との差
$2^{32}-1$	OK	
$2^{31}-1$	OK	
$2^{30}-1$	OK	
	⋮	

Tbl 1 を入力数値 $n \in \{0\} \cup \mathbb{Z}_+$ に対して，アセンブルの可能条件を一般化する．

規則 1

$$2^p < n \leq 2^{p+1} \qquad (p = 2k+6)$$

となる $k(1 \leq k \leq 9)$ に対して，

$$n \bmod 4^k = 0$$

が n をアセンブルできる条件である．

[*2] 実行するに際して，空ファイル test.s を作成し，mov.s の一部を変更する必要がある．

1.4　考察

■結果からの考察　Tbl 1 より，mov 命令による 24 ビットまでの即値の代入に関しては**規則 1** に従っていることが判る．さらに，入力数値のビット列が 8 ビットに収まり，その 8 ビットに収まったビット列の先頭と，レジスタの 1 ビット目との距離が 2 の倍数である必要がある（Fig 1）．ただし，Tbl 2 より 24 ビット以降の即値の代入に関してはこの限りでない．理由は不明である．

Fig 1　アセンブル可能な即値の例

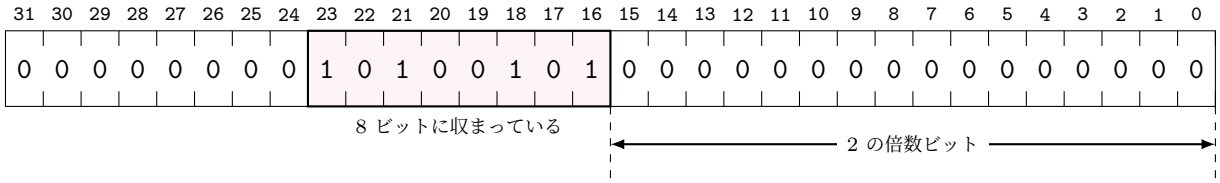
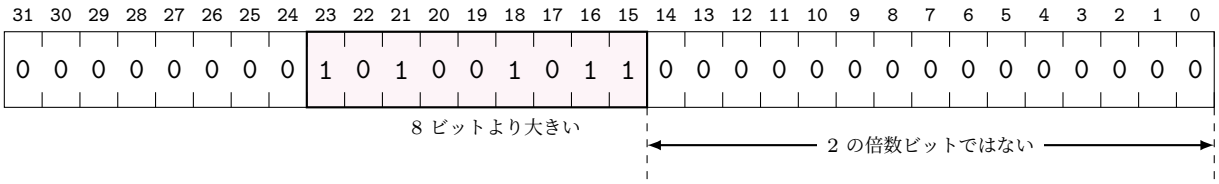


Fig 2　アセンブルが不可能な即値の例

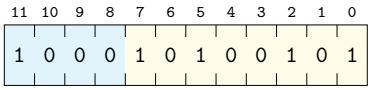


■arm Assembler の mov 命令の詳細　資料 [1] によると, ARM データの処理命令ビットレイアウトで，即値に当てられているのは下位 12 ビットのみである．

さらに，12 ビットの即値を 12 ビットの番号として処理するのではなく，4 ビットの回転値と 8 ビットの値として処理する（Fig 3）．この回転値 r は即値 im を右に $2r$ ローテートするために用いる．

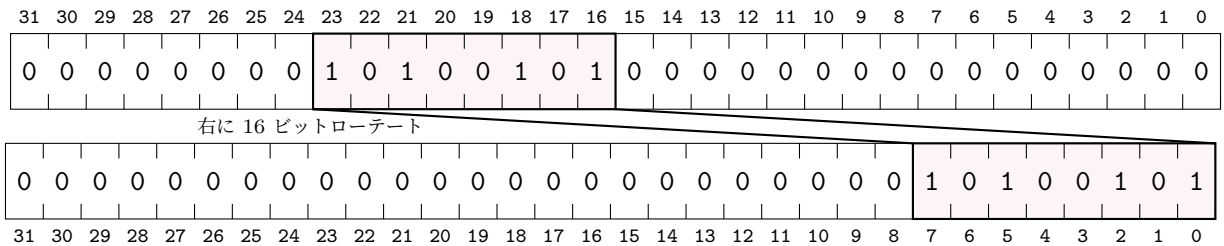
Fig 3　即値の利用

Fig 4　結果



仮に $i =$ 0x00D30000 が入力されたとする．i のビット列と右ローテートを Fig 5 に示す．i はそのままだと即値として利用できないため, 即値として利用できる下位 8 ビット im にローテートさせる．（シフトではない．）このとき 16 ビットシフトさせたのでこの値を 2 で割った 8 を回転数 r に格納し，下位 12 ビットのみで 23 ビット必要である数値 $i =$ 0x00D30000 を Fig 4 のように表すことができる．

Fig 5　i のビット列，右ローテート



■このような設計になっている理由　ARM データ処理命令のビットレイアウト [1] によると，1 つのレジスタ内に Cond や，加減算，移動，比較などを正確に定義するための値も格納されている．開発者は即値に 32 ビットを割くよりも，正確さを保証し 12 ビットでより多くの即値を表現すると考えたのではないだろうか．

参考文献

[1] alisdair mcdiarmid. *arm immediate balue encoding*.
https://alisdair.mcdiarmid.org/arm-immediate-value-encoding/.