コンピュータアーキテクチャII

アセンブリ言語・レポート

k19100

松永一希

REPORT

No. 1

課題：高水準言語の1命令は低水準言語の複数命令に対応することを調べる。
(fibonacci.c) (fibonacci.s)
フィボナッチ数列を表示するプログラムC版，x86アセンブラ版を用いて，
その対応関係を説明しなさい。

○C言語

① int a=1, b=1, n, tmp; …(1)(2)

①変数の作成
(1) C言語ではint(32bit)の変数を作成し，aとbの変数では1を代入し，n，tmpでは何も代入せず，変数だけ作成。
アセンブリ言語では，long(32bit)のaというラベルを作成し，1を代入。
これにより，C言語でのintはアセンブリ言語ではlongとなる。
(2) .commについてリファレンスは .comm name, size, alignmentとなるため，変数名はnとし，intは32bitすなわち4byteで，サイズは4倍のため参照位置は4となる。(1)

② int main(void){…}

②メイン文での処理
アセンブリ言語ではディレクティブを使用し，メイン文を表す。

③ n=0;

③値を代入
アセンブリ言語ではmovlを使用して値を代入。$マークを付けることで即値となる。

④ printf("%10d\n", b);

○アセンブリ言語

```
① a: .long 1          …(1)
   b: .long 1
      .comm n, 4, 4    …(2)
      .comm tmp, 4, 4

② .text
   .globl main
main: …

③ movl $0, n          # n=0

④ fmt1: .string "%10d\n"  …(1)
main:
      movl b, %esi          …(2)
      movl $fmt1, %edi      …(3)
      movl $0, %eax         …(4)
      call printf           …(5)
```

REPORT

No. 2

④ 文字列に変数の値を代入して表示させる。

(1) fmt1 というラベルに .string オペコードを入れ，"%10d\n" という文字列のオペランドを入れる。

(2) b という変数を %esi というレジスタに代入する。

(3) fmt1 ラベルを %edi レジスタに代入する。

(4) %eax レジスタに 0 を代入する。

(5) gcc を呼んで (1)～(4) までの流れを表示させる。

⑤ while((tmp = a+b)>0){…}

⑤
```
while:
        movl    a, %eax
        addl    b, %eax     …(1)
        movl    %eax, tmp
        testl   %eax, %eax  …(2)
        jle     endwhile    …(3)
```

⑤ while文の作成

(1) addl というオペコードは，元にあったレジスタの値を新しく代入した値と足し，レジスタに代入する。

(2) %eax レジスタをテスト（実行）する。

(3) もし，(2)でテストした値が 0以下になった場合，endwhile というラベルへ移動する。

⑥
```
b = a;
a = tmp;
```

⑥
```
movl    a, %eax     …(1)
movl    %eax, b     …(2)
movl    tmp, %eax
movl    %eax, a     …(3)
```

⑥ 変数の値に別の変数の値を代入する。

(1) %eax レジスタに一時的に a の値を代入する。

(2) b に %eax レジスタの値を代入する。

⑦
```
if(a>1000){
    n++;    …(3)
}
```

⑦
```
cmpl    $1000, %eax …(1)
jle     endif       …(2)
movl    n, %ebx
addl    $1, %ebx    …(3)
movl    %ebx, n
```

REPORT

No. 3

⑦ if文の作成

(1) cmpl というオペコードは，オペランド同士を比較する。⑥-(3)で %eax レジスタに代入した a の値と 1000 を比較する。

(2) もし，1000 が %eax 以上だった場合，endif というラベルに移動する。

⑧ printf("%10d\n", a);
} // while文の閉じ{}

```
endif:    …(1)
    movl  a, %esi
    movl  $fmt1, %edi
    movl  $0, %eax
    call  printf
    jmp   while   …(2)
```

⑧ if文の処理後

(1) アセンブリ言語は最低限の仕組みしかないため，if文の終了するラベルも作成する必要がある。

(2) 処理後，while文に戻る。

⑨ printf("\nThe number …", n);
printf("Final values …", a, b, tmp);
exit(0);

```
fmt2: .string "\nThe number …"
fmt3: .string "Final values …"
endwhile:
    movl  n, %esi
    movl  $fmt2, %edi
    movl  $0, %eax
    call  printf

    movl  tmp, %ecx
    movl  b, %edx      …(1)
    movl  a, %esi
    movl  $fmt3, %edi
    movl  $0, %eax
    call  printf

    movl  $0, %edi     …(2)
    call  exit         …(3)
```

REPORT

No. 4

⑨ while文の処理後とプログラムの終了

いい tmp, b, a をそれぞれ %ecx, %edx, %esi レジスタに代入している。

ええ %edi に0を代入することで処理を終了。

おい プログラムを終了。

%esi, %ecx, %edx などそれぞれレジスタを参照するための名前を定義できることが分かった。

また, オペコードは様々なオペランドをとる必要があるため, 型の表記は決まっている。

参照文献：[1] ja.wikibooks.org/wiki/X86アセンブラ/x86アーキテクチャ (検索日 11月3日)