アセンブリ言語例題解説

問

以下に示すのは、10 個の整数を入力として読み込んで、そのうちで偶数であるものだけを出力する MIPS アセンブリ言語プログラムである。スタックフレームに関する操作及び、main の呼び出し元に戻るための処理は簡単のため省略してある。参考のため、同じことを行なう C 言語のプログラムを左に並べて示した。

プログラム中の 6 か所の空欄ア〜カを適切なアセンブリ言語表現で埋めなさい。なお、正答は一通りとは限らない。

C のプログラム

```
#include <stdio.h>

int main()
{
  int i;
  int n;
  int r;

  for (i = 0; i < 10; i++) {
    scanf("%d", &n);
    r = n % 2;
    if (r == 0) {
      printf("%d\n", n);
    }
  }
  return 0;
}
```

アセンブリ言語プログラム

```
        .data
str:    .asciiz "\n"
        .text
        .globl      main
main:   li          [ ア ], 0
loop:   [ イ ]      $t0, 10, end
        li          $v0, 5
        syscall
        move        $t1, [ ウ ]
        rem         $t2, $t1, 2
        [ エ ]      $t2, $0, skip
        move        $a0, [ オ ]
        li          $v0, 1
        syscall
        la          $a0, str
        li          $v0, 4
        syscall
skip:   addiu       $t0, [ カ ], 1
        b           loop
end:
```

ヒント 1: MIPS のアセンブリ言語命令 (または疑似命令) の一部を以下に示す:

`li Rdest, Imm`	即値 `Imm` を `Rdest` にロード
`move Rdest, Rsrc`	`Rsrc` の内容を `Rdest` に入れる
`addiu Rdest, Rsrc1, Imm`	`Rsrc1` と即値 `Imm` の和を `Rdest` へ
`rem Rdest, Rsrc1, Src2`	`Rsrc1` を `Src2` で割った余りを `Rdest` へ
`la Rdest, label`	`label` が指すアドレスを `Rdest` へ
`b label`	無条件分岐
`beq Rs, Rt, label`	`Rs` と `Rt` が等しい時に分岐
`bne Rs, Rt, label`	`Rs` と `Rt` が等しくない時に分岐
`bge Rs, Rt, label`	`Rs` $\geq$ `Rt` の時に分岐
`bgt Rs, Rt, label`	`Rs` $>$ `Rt` の時に分岐
`ble Rs, Rt, label`	`Rs` $\leq$ `Rt` の時に分岐
`blt Rs, Rt, label`	`Rs` $<$ `Rt` の時に分岐
`syscall`	システムコール (番号は`$v0`, 引数は`$a0`, 返値は `$v0`)
	システムコール 1 番は `int` の印刷
	システムコール 4 番は文字列の印刷

ヒント 2: 0 番レジスタ $0 には常に 0 が入っている。

解法

C のプログラムを見ると、i, n, r の 3 つの変数がある。変数の値をどこにしまっているかを考えるのが一つのキーポイント。値はメモリにしまっても良いが、レジスタのほうが高速にアクセスできる。そこで変数はレジスタに割り付けられることが多いから、レジスタを使っている命令に着目しよう。

アセンブリプログラムを見ると `rem $t2, $t1, 2` という命令があるので、これに着目してみる。ヒントを見ると、これは `$t1` を 2 で割った余りを `$t2` に入れる命令である。これを C のプログラムと見比べると、`r = n % 2` (n を 2 で割った割りを r に格納) に相当することがわかる。従って、`r` が `$t2` に、`n` が `$t1` にあたることがわかる。

次に手がかりになりそうなのは、for 文のループ継続条件 `i < 10` である。10 という数値は他に出てこないから、アセンブリプログラムに 10 が出てきたら、このループ継続条件に対応する部分だと推測できる。`loop` というラベルがついている命令がそれである。`$t0` と 10 を比較してその結果によって `end` というラベルに分岐するかどうかを決めているようだ。従って、`$t0` が `i` だと考えられる。これで C の変数とレジスタの対応がわかった。ところで、`for` 文のループ継続条件の判定においては、「条件が成り立たなくなったらループの末尾に飛んでループを終わる」と考えれば良い、ということを前に説明した。つまり、`i >= 10` になったら`end` に飛べばよい。従って、`$t0` の値 ≥ 10 のとき分岐すればよいから、[イ]の欄には条件分岐 `bge` が入る。

あとは頭から見て行こう。C プログラムで最初に実行されるのは、`for` 文の初期化部 `i=0` である。`i` は `$t0` に割り当てられていたから、アセンブリコードでは、`$t0` に即値 0 を入れればよい。従って、[ア]は`$t0` である。

`scanf` に対応する入力処理をみよう。5 番のシステムコールで int を入力しているが、その返値は (ヒントを見ると)`$v0` に入る。`scanf` では変数 `n` に入力を読み込んでいるので、アセンブリコードでは`$t1` に入力値を入れるべきである。従って、`$v0` から`$t1` に値をコピーすべきだから、[ウ]欄には`$v0` が入る。

次に `if` 文を見る。`r = n % 2` の直後が `if` 文だから、アセンブリコードでは rem 命令の直後と考えられる。`$t2` と 0 を比べた結果によりラベル `skip` を飛ぶかどうかを決めていると考えられる。`r=0` が成立しないときに `r` の印刷処理を飛び越せばいいから、[エ]に入るのは `bne` である。

printf 文を見る。n の値を印刷するので、システムコールの引数として n の値、すなわち $t1 の値を $a0 (システムコールの引数を入れるレジスタ) に入れればよい。従って、［ オ ］は$t1 である。このあともう一回印刷のためのシステムコールをしているが、2 回目は改行 ("\n") を印刷しているだけである。

if 文の処理が終ったら、for 文の i++ に相当する処理をしてからループの先頭に分岐する。i++ は i=i+1 と考えられるので、アセンブリコードでは、$t0 の値に 1 を加算してその結果をまた $t0 に入れればよい。従って、［ ウ ］は$t0 である。