マイクロプロセッサ演習

2004 年度

第 5 回

1 [SPIM] if 文

C 言語における if 文は図 1 のように実現される。“条件” が成立する時だけ、if(条件){ } の中の命令

```
if(条件){
        命令;
}
```

図 1: C 言語における if 文

が実行される。これは MIPS のアセンブリ言語では図 2 のように実現される。

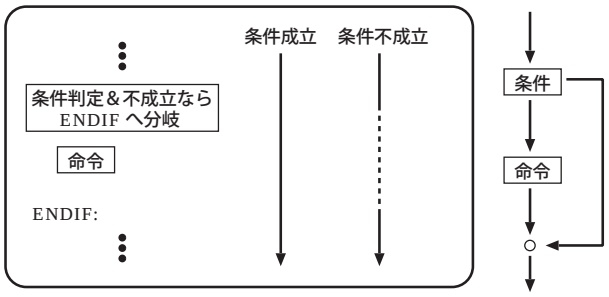

図 2: MIPS のアセンブリ言語における if 文

前回の while 文もそうであったが、MIPS のアセンブリ言語には if 文に直接対応するものがあるわけではなく、条件命令や分岐命令 (`beq`、`bne`、`slt`、`j` など) とラベルを組み合わせることで if 文を実現している。なお、図中の “ENDIF” は単なるラベルであり、好きな名前をつけて構わない。

本章で実際に if 文を書いてもらう。演習の Web ページより、data05.zip をダウンロードする。この中に、今日の演習で使うプログラムの雛形が入っている。If.asm がこの問題で作成してもらうプログラムの雛型である。

メモリ上に array[0]、array[1] が確保されており、ともに 2 が代入されている。また、msg というラベルで “equal.” という文字列が格納されている。

このファイルに追加する形で、「array[0] = array[1] ならば msg をコンソールに出力。array[0] ≠ array[1] なら何もしない。」というプログラムを作成せよ。なお、msg を画面に出力するには、

```
li      $v0, 4
la      $a0, msg
syscall
```

を記述すること。

また、array[0] と array[1] の値をいろいろ変えて保存& 実行し、分岐が正しく行なわれることを確認せよ。

2 [SPIM] if-else 文

C 言語における if-else 文は 図 3 のように実現される。“条件” が成立する時は命令 1 が、成立しな

```
if(条件){
        命令1;
}else{
        命令2;
}
```

図 3: C 言語における if-else 文

い時は命令 2 が実行される。これを MIPS のアセンブリ言語で実現するには図 4 のようにする。本章ではこの if-else 文を記述してもらう。プログラムの雛型は IfElse.asm である。

メモリ上に array[0]、array[1] が確保されており、ともに 2 が代入されている。また、msg1 というラベルで “equal.” という文字列が、msg2 というラベルで、“not equal.” という文字列が格納されている。

このファイルに追加する形で、「array[0] = array[1] ならば msg1 をコンソールに出力。array[0]

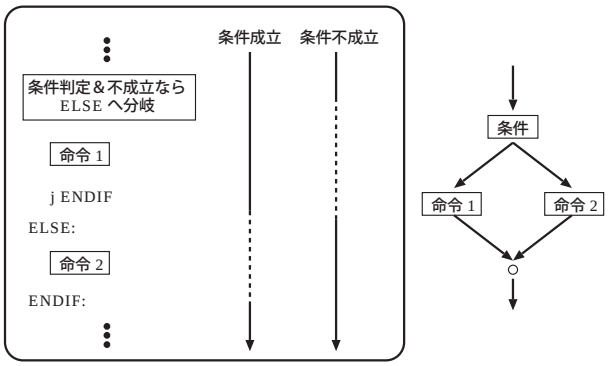

図 4: MIPS のアセンブリ言語における if-else 文

≠ array[1] なら msg2 をコンソールに出力。」というプログラムを作成せよ。

また、array[0] と array[1] の値をいろいろ変えて、分岐が正しく行なわれていることを確認せよ。

3 [SPIM] switch 文

C 言語における switch 文は、図 5 のように実現される。$k = 0, 1, 2, \cdots$ の値に応じて、それぞれ命

```
switch(k){
    case 0:
        命令 0;
    case 1:
        命令 1;
    case 2:
        命令 2;
    ...
}
```

図 5: C 言語における switch 文

令 0、命令 1、命令 2、$\cdots$ が実行さる。

これを MIPS のアセンブリ言語で実現するには図 6 のようにする。基本的には while 文、if 文、if-else 文と同様に、条件判定と分岐命令を組み合わせることで実現するが、switch 文の場合「ジャンプ・アドレス表」という概念が必要になるので図 7 を用いてそれを解説する。

図 7 は switch 文を使ったプログラムを実行したときのメモリ上のデータの状態の一例である。メモリには「**テキスト 領域**」と「**データ領域**」があり、**テキスト 領域**には、実際のプログラムの命令列が、

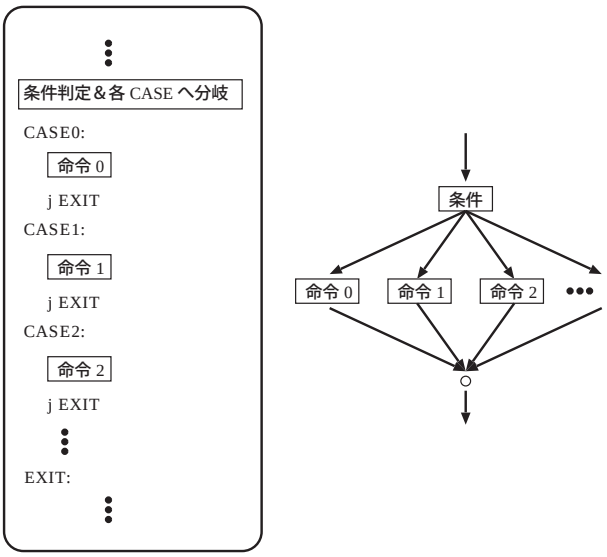

図 6: MIPS のアセンブリ言語における switch 文

データ領域には、プログラムで用いられるデータがそれぞれ格納される。前回までのメモリの模式図には、データ領域のみが書かれていたことに注意しよう。

さて、例えばプログラムを CASE0 に分岐させたい時はどうすればよいだろうか。テキスト領域を見ると、CASE0 はメモリアドレス 0x00400088 に対応している。だから、もしレジスタ $t0 に 0x00400088 が代入されていれば、

```
jr $t0
```

という命令で CASE0 に分岐することができる。

同様に CASE1、 CASE2、 CASE3 はそれぞれ 0x0040009c、0x004000b9、0x004000c4 のアドレスに対応している。これらのアドレスを表のように並べたものを、ジャンプ・アドレス表という。図中では case[0]、case[1]、case[2]、case[3] に各ラベルのアドレスが書かれており、これがジャンプ・アドレス表である。この表から、ジャンプすべきアドレスをレジスタに読み出して `jr` 命令を実行すれば、switch 文が実現されることになる。

本章では、実際に switch 文を自分で記述してもらう。プログラムの雛型は Switch.asm である。

メモリ上に k という変数が確保されており、まず、その値が $k < 0$ または $k \geq 4$ ならば、EXIT に分岐させ、その後、$k = 0, 1, 2, 3$ であればそれぞれ CASE0、CASE1、CASE2、CASE3 に分岐させる。

分岐させた後は画面に文字列を表示させるのだ

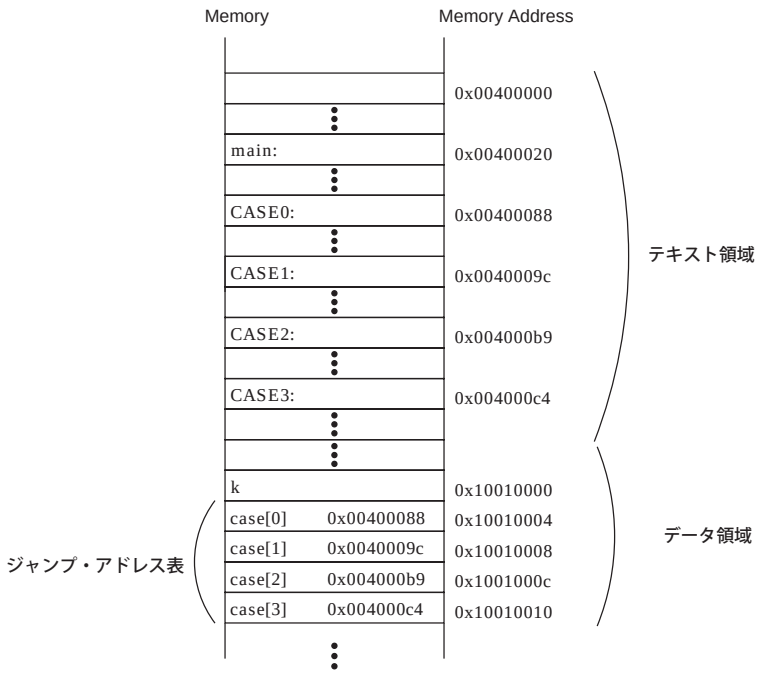

図 7: switch 文を実現するために用いるジャンプ・アドレス表

が、この部分はすでに書かれている。また、ジャンプアドレス表も既に作成されているので、分岐構造の部分のみを記述すること。

記述ができたら、k の値をいろいろ変え、正しく分岐が行なわれていることを確かめて欲しい。

なお、この問題は授業のプリントや教科書に基づいているので、そちらがヒントになるかもしれない。