MIPS 生命游戏实验报告

> 数据科学与计算机学院 2017级 软件工程 5 班 梁文杰 16303050

一、实验目的

1. 熟悉常用的MIPS汇编指令/能够阅读调试以及编写程序；
2. 了解MIPS处理器结构以及寄存器功能；
3. 学会MIPS程序编写环境（如PCspim）的使用和调试技术；
4. 掌握MIPS汇编指令分别和机器语言和C语言语句之间的对应关系，掌握程序的内存映射。

二、实验内容

用 MIPS 代码模拟一个生命游戏的过程。

> **生命游戏介绍**
>
> 康威生命游戏（英语：Conway's Game of Life），又称康威生命棋，是英国数学家约翰·何顿·康威在1970年发明的细胞自动机。
>
> 生命游戏中，对于任意细胞，规则如下：
>
> 1. 每个细胞有两种状态：存活或死亡，每个细胞与以自身为中心的周围八格细胞产生互动。
> 2. 当前细胞为存活状态时，当周围的存活细胞低于2个时（不包含2个），该细胞变成死亡状态。（模拟生命数量稀少）
> 3. 当前细胞为存活状态时，当周围有2个或3个存活细胞时，该细胞保持原样。
> 4. 当前细胞为存活状态时，当周围有超过3个存活细胞时，该细胞变成死亡状态。（模拟生命数量过多）
> 5. 当前细胞为死亡状态时，当周围有3个存活细胞时，该细胞变成存活状态。（模拟繁殖）
>
> 可以把最初的细胞结构定义为种子，当所有在种子中的细胞同时被以上规则处理后，可以得到第一代细胞图。规则继续处理当前的细胞图，可以得到下一代的细胞图，周而复始。

用户通过键盘输入一个迭代次数 n，根据给定的初始细胞图进行 n 次的迭代，返回每次迭代后的细胞图状态。

三、设计思路与分析

先用 c++ 语言实现程序功能，进而用 MIPS 语言进行翻译。

本实验设计为一块 15*15 的细胞板，分别用 N 和 Nsquare 来存储细胞板的宽度和细胞总数，newBoard 及 board 为每一位细胞申请 1bit 的空间储存当前的存活状态。

实际上程序分成了三个部分：循环遍历部分、求周围存活细胞数、打印输出当前状态。根据每个部分各自的逻辑完成相应的代码。

主函数进行n次迭代，每次迭代遍历所有的细胞，计算出当前细胞周围存活的细胞数，根据函数返回值进行判断改细胞在下一个状态是否仍然存活，或是否有死亡状态变为存活状态等，遍历完后将表格中的每个细胞状态打印出来，并完成细胞板的转移，供下一次迭代使用。

c++ 代码：

```cpp
#include <bits/stdc++.h>
using namespace std;
#define forto(i,b,n) for(int(i)=(b);(i)<=(n);++(i))

int max_iters;
const int SIZE=15;
char newboard[SIZE][SIZE];
char board[SIZE][SIZE] = {
   {1,0,0,0,0,0,0,0,0,0,1,0,0,0,0},
   {1,1,0,0,0,0,0,0,0,0,1,1,0,0,0},
   {0,0,0,1,0,0,0,0,0,0,1,0,1,0,0},
   {0,0,1,0,1,0,0,0,0,0,1,0,0,1,0},
   {0,0,0,0,1,0,0,0,0,0,1,0,0,0,1},
   {0,0,0,0,1,1,1,0,0,0,1,0,0,1,0},
   {0,0,0,1,0,0,1,0,0,0,1,0,1,0,0},
   {0,0,1,0,0,0,1,0,0,0,1,1,0,0,0},
   {0,0,1,0,0,0,1,0,0,0,1,0,1,0,0},
   {0,0,1,0,0,0,1,0,0,0,1,0,0,1,0},
   {0,0,1,0,0,0,0,0,0,0,1,0,0,0,1},
   {0,0,1,0,0,0,0,0,0,0,1,0,0,1,0},
   {0,0,1,0,0,0,0,0,0,0,1,0,1,0,0},
   {0,0,1,0,0,0,0,0,0,0,1,1,0,0,0},
   {0,0,1,0,0,0,0,0,0,0,1,0,0,0,0},
};

int neighbours(int i,int j){
   int cnt=0;
   forto(x,-1,1)forto(y,-1,1)
     if(i+x<0||i+x>SIZE-1||j+y<0||j+y>SIZE-1||(x==0&&y==0))continue;
     else cnt+=(board[i+x][j+y]==1);
   return cnt;
}

void copyBackAndShow() {
  int i,j;
  forto(i,0,SIZE-1){
    forto(j,0,SIZE-1){
      board[i][j]=newboard[i][j];
      putchar(board[i][j]==0?'.':'#');
    }
    putchar('\n');
  }
}

int main(void) {
  printf("# Iterations: "),cin>>max_iters;
  forto(n,1,max_iters){
    forto(i,0,SIZE-1){
      forto(j,0,SIZE-1){
        int cnt=neighbours(i,j);
        if(cnt<2)newboard[i][j]=0;
        else if(cnt==2)newboard[i][j]=board[i][j];
        else if(cnt==3)newboard[i][j]=1;
```

```
        else newboard[i][j]=0;
      }
    }
    printf("\nAfter Iteration %d\n", n);
    copyBackAndShow();
  }
  return 0;
}
```

翻译成 MIPS 汇编为:

变量定义:

```
.data
  N: .word 15
  Nsquare: .word 224
  newBoard: .space 225
  iterations: .word 0
  hash: .asciiz "#"
  dot: .asciiz "."
  inputMessage: .asciiz "# Iterations: "
  afterIteration: .asciiz "\nAfter iteration "
  endl: .asciiz "\n"
  board:
   .byte 1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0
   .byte 1,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0
   .byte 0,0,0,1,0,0,0,0,0,0,0,1,0,1,0,0
   .byte 0,0,1,0,1,0,0,0,0,0,0,1,0,0,1,0
   .byte 0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,1
   .byte 0,0,0,0,1,1,1,0,0,0,0,1,0,0,1,0
   .byte 0,0,0,1,0,0,1,0,0,0,0,1,0,1,0,0
   .byte 0,0,1,0,0,0,1,0,0,0,0,1,1,0,0,0
   .byte 0,0,1,0,0,0,1,0,0,0,0,1,0,1,0,0
   .byte 0,0,1,0,0,0,1,0,0,0,0,1,0,0,1,0
   .byte 0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,1
   .byte 0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0
   .byte 0,0,1,0,0,0,0,0,0,0,0,1,0,1,0,0
   .byte 0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0
   .byte 0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0
```

主函数部分:

```
.text
.globl main
main:
  la $a0,inputMessage                # Print message
  li $v0,4
  syscall
  li $v0,5                           # Read user input
  syscall
  la $v1,iterations
  sw $v0,0($v1)                      # Save iterations
```

```
initial:
  li $s0,1                              # Set iterations counter to 1
  iterations_loop:                      # Cycling loop begins
    lw  $t0,iterations                  # Load iterations
    bgt $s0,$t0,end_iterations_loop     # Compare iter and counter
    li  $s1,0                           # Initial counter $s1
    logic_loop:                         # loop from 0 to Nsquare
      lw   $t0,Nsquare                  # Get exit status
      bgt  $s1,$t0 logic_end            # Exit if counter > Nsquare
      move $a0,$s1                      # $a0 = current index
      jal  neighbours                   # Get neighbours
      move $s3,$v0                      # Copy the amount of neighbours
      li   $t0,2
      blt  $s3,$t0,dead
      beq  $s3,$t0,keep
      li   $t0,3
      beq  $s3,$t0,alive
      dead:
        sb $zero,newBoard($s1)          # Write 0 to newBoard
        j  end_incr_j                   # end loop iteration
      alive:
        li $t0,1
        sb $t0,newBoard($s1)            # Write 1 to newBoard
        j  end_incr_j                   # end loop iteration
      keep:
        lb $t0,board($s1)               # Load the status of current cell
        sb $t0,newBoard($s1)
    end_incr_j:
      addi $s1,1                        # End inner loop
      j logic_loop
  logic_end:
    la $a0,afterIteration               # Print "After iteration "
    li $v0,4
    syscall
    move $a0,$s0                        # Print the iteration number
    li $v0,1
    syscall
    la $a0,endl
    li $v0,4
    syscall
    jal copyBackAndShow                 # Print and copy
    addi $s0,1                          # Increment and save back
    j iterations_loop                   # Next iteration
  end_iterations_loop:
exit:
  li $v0, 10
  syscall                               # exit
```

求周围存活细胞数函数:

```
neighbours:
  li  $v0,0                             # Initial counter of neighbours $v0
  lw  $t9,N                             # Load constant N
```

```
  # Cartesian coordinates: $t0 = $t5*N + $t6
  div $t5,$a0,$t9                       # $t5 = integerDivision $t0/N
  rem $t6,$a0,$t9                       # $t6 = mod  $t0 % N
  li  $t1,-1                            # Init counter $t1
  li  $t2,-1                            # Init counter $t2
  li  $t8,1                             # Load branching constant into $t8
  outerNeighbours:
    bgt $t1,$t8,outerNeighboursEnd      # Jump to the end if counter greater than 1
    li  $t2,-1                          # Reset counter
    innerNeighbours:
      bgt  $t2,$t8,innerNeighboursEnd   # Jump to the end if counter greater than 1
      add  $t7,$t1,$t5                  # Border check
      bltz $t7,caseFail                 # above board if less than 0
      bge  $t7,$t9,caseFail             # below board if greater than N-1
      add  $t3,$t2,$t6                  # Border check
      bltz $t3,caseFail                 # above board if less than 0
      bge  $t3,$t9,caseFail             # below board if greater than N-1
      mul  $t7,$t7,$t9                  # +(-N||0||N)
      add  $t7,$t7,$t3
      beq  $t7,$a0,caseFail             # Pass index itself
      lb   $t4,board($t7)               # Add the board index $t3 into $t4
      add  $v0,$v0,$t4
    caseFail:
      addi $t2,$t2,1                    # Incement and jump
      j innerNeighbours
    innerNeighboursEnd:
      addi $t1,$t1,1                    # Increment outer counter and jump to start of
loop
      j outerNeighbours
  outerNeighboursEnd:
    jr $ra
```

打印状态及拷贝函数:

```
copyBackAndShow:
  lw $t0,N                              # $t0 = N
  li $t1,0                              # Init counter $t1
  li $t2,0                              # Init counter $t2
  outerCopy:
    beq $t1,$t0,endOuterCopy            # Exit if outer counter equals N
    li $t2,0                            # Reset inner counter
    innerCopy:
      beq $t2,$t0,endInnerCopy          # Jump if done
      mul $t3,$t1,$t0                   # Get offset into $t3,: offset = $t1*N + $t2
      addu $t3,$t3,$t2
      lb $t5,newBoard($t3)              # Copy newBoard to board
      sb $t5,board($t3)
      lb $t4,board($t3)
      beqz $t4,caseDot                  # Determine print hash or dot
        la $a0,hash                     # Print hash
        li $v0,4
        syscall
        addi $t2,$t2,1                  # Increment and jump to the start of the loop
```

```
        j innerCopy
      caseDot:
        la $a0,dot                    # Print dot
        li $v0,4
        syscall
        addu $t2,$t2,1                # Increment and jump to the start of the loop
        j innerCopy
    endInnerCopy:
      la $a0,endl                     # Print endl
      li $v0,4
      syscall
      addu $t1,$t1,1                  # Increment row counter and continue
      j outerCopy
  endOuterCopy:
    jr $ra
```

四、实验效果

五次迭代后的结果：

```
# Iterations: 5

After Iteration 1
##........##...
##.......##....
.###.....##.#..
....#....##..#.
....#....###.##
...##.#..##..#.
...##.##.##.#..
..##.###.##.#..
.###.###.##.#..
.###.....##..#.
.###.....###.##
.###.....##..#.
.###.....##.#..
.###.....##....
..........##...

After Iteration 2
##.......###...
...............
####....#......
..#.#...#....##
....#...#..#.##
......##.....##
............##.
.#..........##.
.....#.#....##.
#.....#......##
#...#...#..#.##
#...#...#....##
#...#...#......
```

```
.#.#...........
..#......###...

After Iteration 3
..........#....
.........##....
.###...........
..#.#..###..###
...#.#..#......
.......#.......
...............
...........#..#
......#........
.....###.......
##...#.#.......
##.###.###..###
##.##..........
.###.....##....
..#.......#....

After Iteration 4
.........##....
..#......##....
.###......#..#.
.#..#..###...#.
...##.#..#...#.
...............
...............
...............
.....###.......
.....#.#.......
###..........#.
...#.#.##....#.
.....#....#..#.
#...#....##....
.###.....##....

After Iteration 5
.........##....
.###.......#...
.#.#......#....
.#..##.####.###
...###.#.#.....
...............
...............
......#........
.....#.#.......
.#...#.#.......
.##.#..##......
.##.#.#.....###
.....##.#.#....
.####......#...
.###.....##....
```

第500迭代后的结果：

```
# Iterations: 500

...

After Iteration 499
........##.....
........##.....
...............
...............
.......##......
......#..#.....
.......#.#.....
........#......
...............
...............
...............
...............
...............
...##..........
...##..........

After Iteration 500
........##.....
........##.....
...............
...............
.......##......
......#..#.....
.......#.#.....
........#......
...............
...............
...............
...............
...............
...##..........
...##..........
```

可以看出细胞图稳定在当前状态。

五、实验心得与体会

在完成这个实验之后对 MIPS 指令的使用算是已经比较熟悉了，能够灵活使用各寄存器及全局变量，通过程序大量的迭代器及跳转掌握了在 MIPS 实现循环数据结构的能力。相比于 x86，MIPS 的指令相对全面很多，比如可以从系统输入直接得到一个整数，而不需要手动进行字符串向整数的转换。

这次实验的另一收获是学会了 MIPS 的函数调用。底层汇编语言的函数调用有三个关键点，一是保存现场，将当前函数域的各种不同变量及 $ra 这样的特殊寄存器保存起来，以免函数返回时出现数据的错误；二是参数传递，三是返回值的传回。前两个主要通过栈来解决，后一个主要通过约定一个寄存器来保存返回值，供调用者获取。