中国科学院大学
University of Chinese Academy of Sciences

计算机科学导论实验

# 编程基础-3

## 斐波那契计算机
## 程序是如何执行的

zxu@ict.ac.cn
zhangjialin@ict.ac.cn

# 编程基础实验安排

<table>
<tr><th>周次</th><th>日期</th><th>实验课</th><th>课程内容</th><th>作业提交内容</th><th>截止时间</th></tr>
<tr><td>2</td><td>3月7日</td><td>编程基础-1</td><td rowspan="2">从hello到Name2Number.go<br>基本数据类型、基本语句、循环</td><td>name_to_number-0.go文件与程序结果</td><td rowspan="3">2025/3/23<br>23:30</td></tr>
<tr><td rowspan="2">3</td><td rowspan="2">3月14日</td><td rowspan="2">编程基础-2</td><td>Name2Number.go文件与程序结果</td></tr>
<tr><td>切片与递归<br>简版快速排序程序<br>fastsort.go</td><td>fastsort.go文件与程序中间结果</td></tr>
<tr><td>4</td><td>3月21日</td><td>编程基础-3</td><td>斐波那契计算机</td><td>RFC</td><td>2025/3/30<br>23:30</td></tr>
</table>

- 作业提交后会立刻返回成绩。
- 在截止时间前四次作业均可反复多次提交。

# 提纲

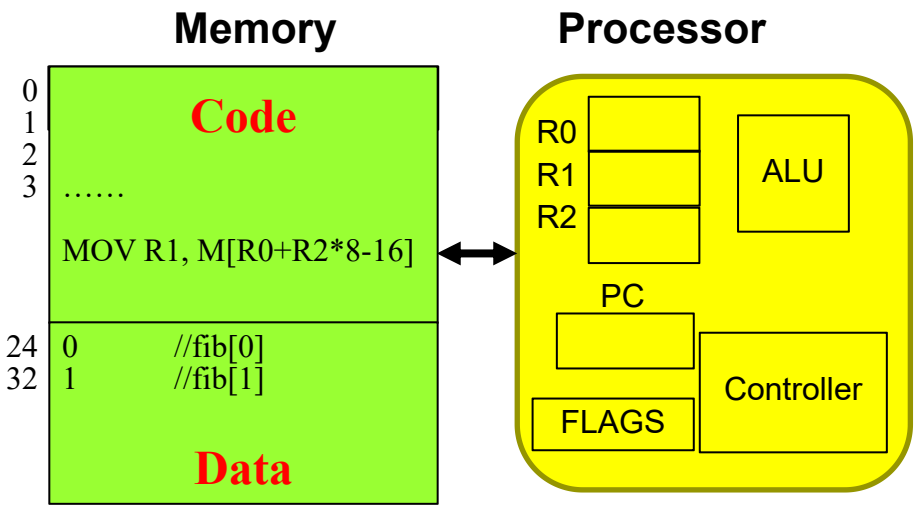
- 目标：理解斐波那契计算机，以及它如何支持数组与循环

- 步骤
  - 复习主课讲授的斐波那契计算机（FC）
    - 组成，指令集，汇编语言程序
    - 逐步执行，导出第13-20步之后的计算机状态
    - 回答关于斐波那契计算机如何支持数组和循环的问题
  - 设计并验证改进型斐波那契计算机（RFC）
    - 假设指令和数据都采用64比特表示，并在原汇编语言程序之后添加一条HALT指令
    - 画出求F(3)的每一步状态，包括从初始状态到执行HALT后的状态
      - 将每一个问号换成数值
  - 提交改进型斐波那契计算机

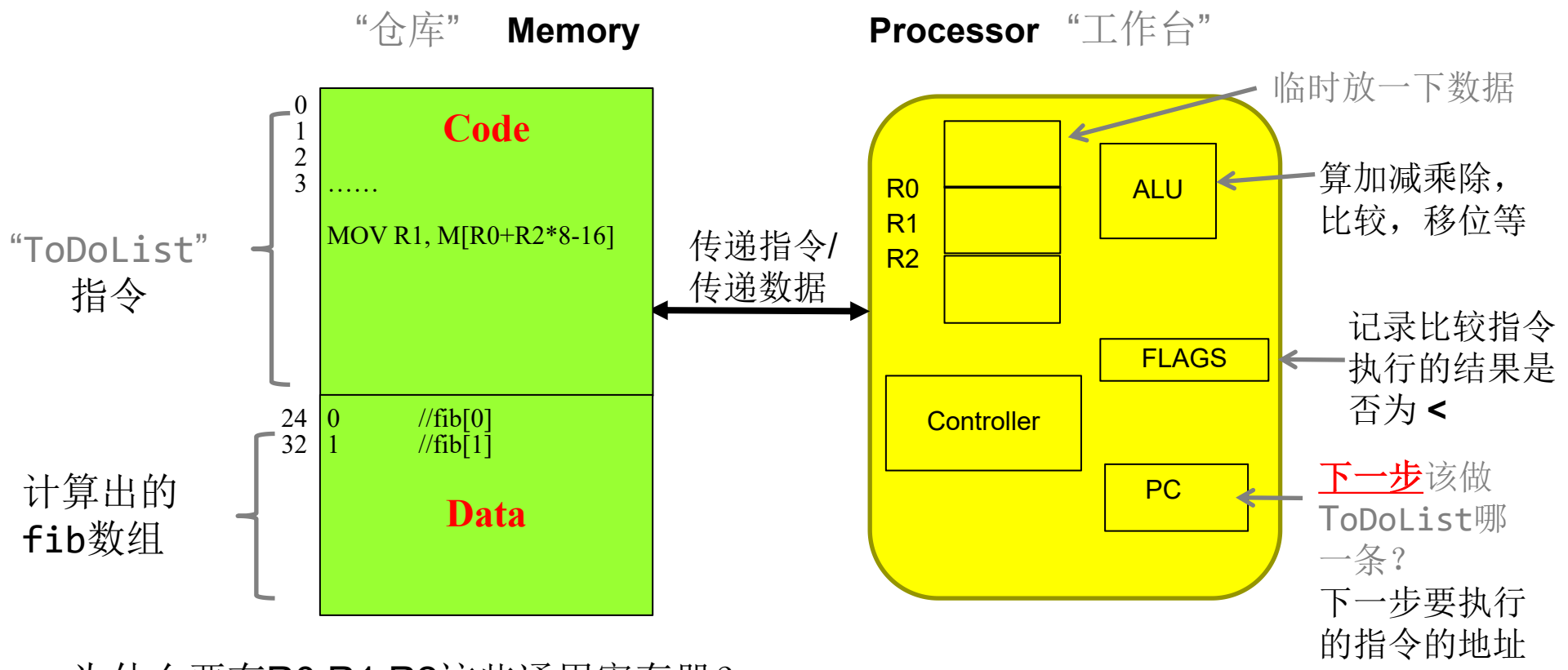
课件中包含教科书未包括的素材引用，特此致谢

# 简化的冯诺依曼模型

- Processer 处理器
  - R0/R1/R2 (Register): 通用寄存器，用来临时存放数据
  - PC (Program Counter): 程序计数器，存储下一条要执行的指令的地址
  - FLAGS: 专用寄存器，存放CMP（比较）指令的结果，是否为 <
  - ALU：运算器
  - Controller: 控制器
- Memory 存储器
  - 字节是访问存储器的基本单位
  - 从0字节开始存放指令（code）
  - 指令后面存放数据（data）



# 简化的冯诺依曼模型

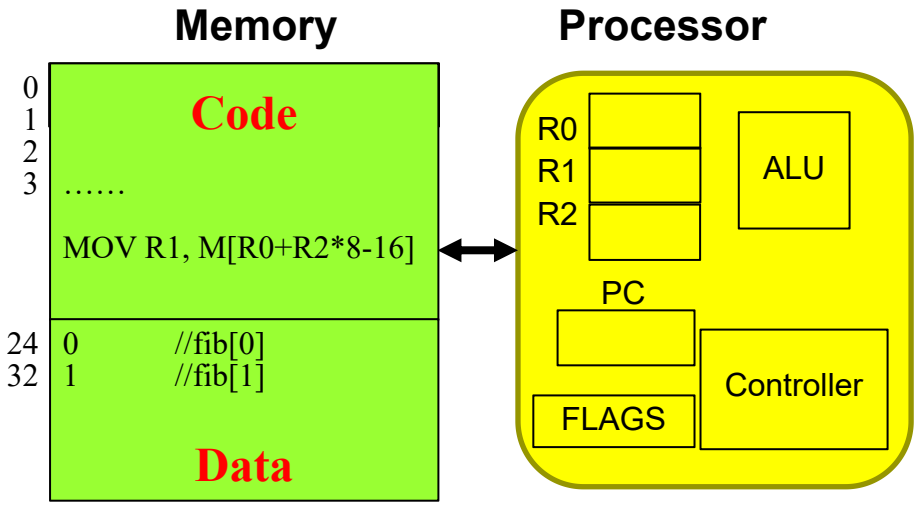


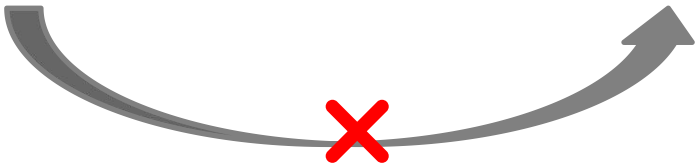

- 为什么要有R0,R1,R2这些通用寄存器？
  每次去“仓库”取太远了（如200个周期），寄存器很快（不到1个周期）等等原因
- Controller用来做什么？
  暂时用人脑代替，模拟执行当前指令，然后将PC改为下一条指令地址

# 简化的冯诺依曼模型

- 人当编译器
  - 将计算 fib[50] 的 go 代码 => 汇编代码
- 一条汇编代码**通常**对应一条机器指令

```
// 计算fib[50]
fib[0] = 0
fib[1] = 1
for i := 2; i < 51; i++ {
    fib[i] = fib[i-1] + fib[i-2]
}
```



go 代码不可以直接执行

go run fib.go也并非直接执行，而是将编译和执行两条命令合在了一起

# 简化的冯诺依曼模型

- 人当编译器
  - 将计算 fib[50] 的 go 代码 => 汇编代码
- 一条汇编代码**通常**对应一条机器指令

```
// 计算fib[50]
fib[0] = 0
fib[1] = 1
for i := 2; i < 51; i++ {
    fib[i] = fib[i-1] + fib[i-2]
}
```

go代码片段

```
MOV 0, R1
MOV R1, M[R0]
MOV 1, R1
MOV R1, M[R0+8]
MOV 2, R2
MOV 0, R1
ADD M[R0+R2*8-16], R1
ADD M[R0+R2*8-8], R1
MOV R1, M[R0+R2*8-0]
INC R2
CMP 51, R2
JL Loop
```

汇编代码片段

# 简化的指令集

- 指令集（共有6条指令）
  - MOV to Register
  - MOV to Memory
  - ADD　　加法指令
  - INC Increment　　增1指令
  - CMP Compare　　比较指令
    小于则置FLAGS为'<'，否则清空FLAGS
  - JL Jump if Less than　　条件跳转

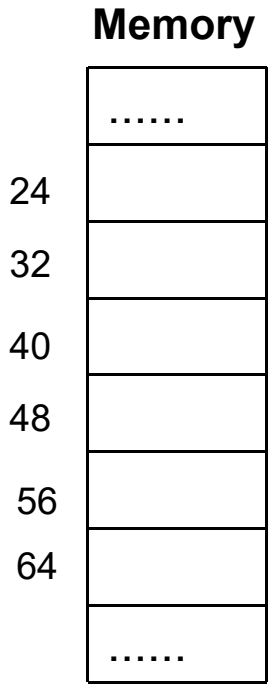
```
MOV 0, R1
MOV R1, M[R0]
MOV 1, R1
MOV R1, M[R0+8]
MOV 2, R2
MOV 0, R1
ADD M[R0+R2*8-16], R1
ADD M[R0+R2*8-8], R1
MOV R1, M[R0+R2*8-0]
INC R2
CMP 51, R2
JL Loop
```

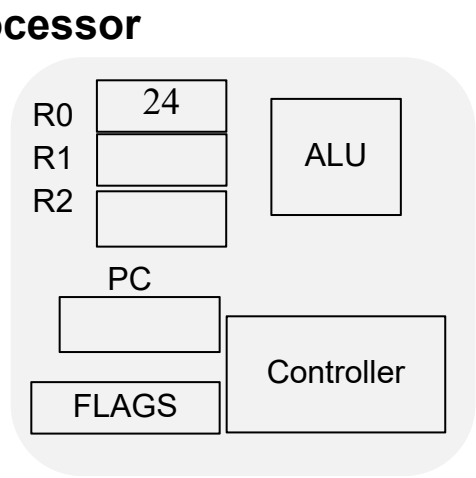
- 一条指令的长度为2个字节(16 bit)
- 一个数据的长度为8个字节(64 bit)

# 指令1：MOV to Register

MOV 0, R1
将 0 存到 R1中

MOV imm, reg
将数字imm存到寄存器reg中



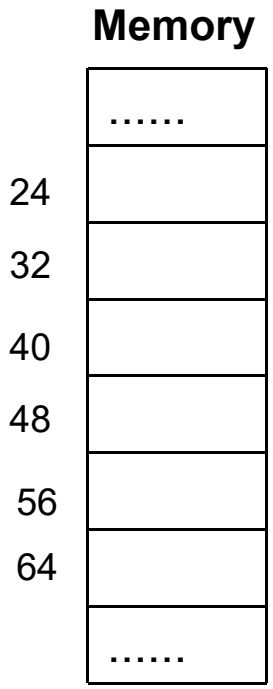



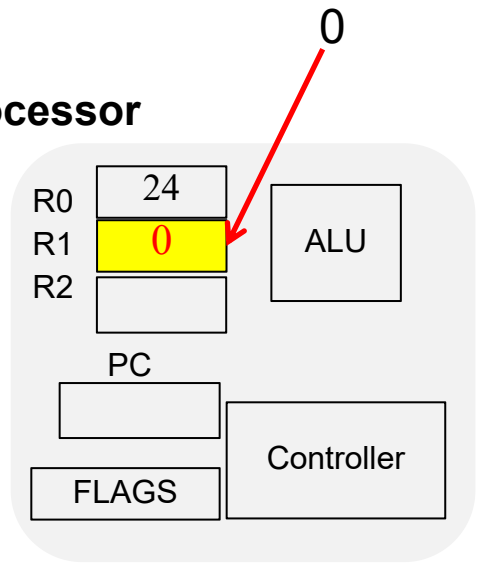

imm: Immediate Value, 立即数
reg: Register，寄存器

# 指令1：MOV to Register

MOV 0, R1
将 0 存到 R1中

MOV imm, reg
将数字imm存到寄存器reg中



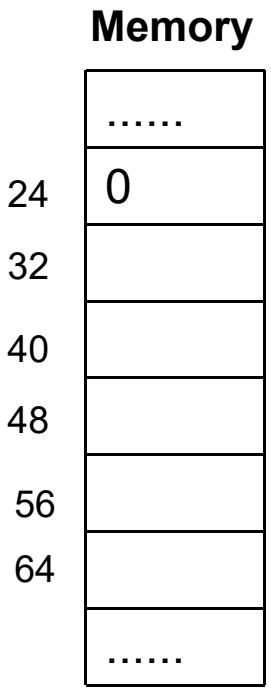



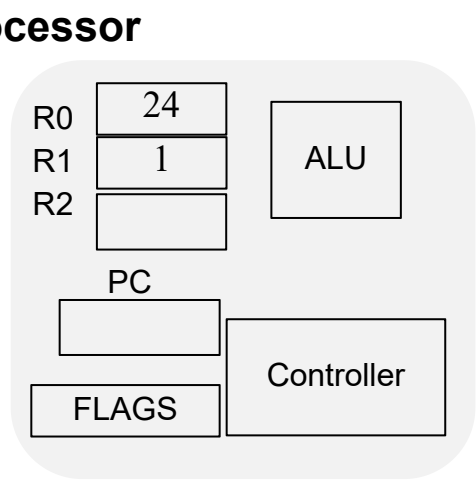

imm: Immediate Value, 立即数
reg: Register，寄存器

# 指令2：MOV to Memory

MOV R1, M[R0+8]
将 R1 中的值存到
M[R0+8] 中

MOV reg, mem
将指定寄存器的值
存到指定内存地址



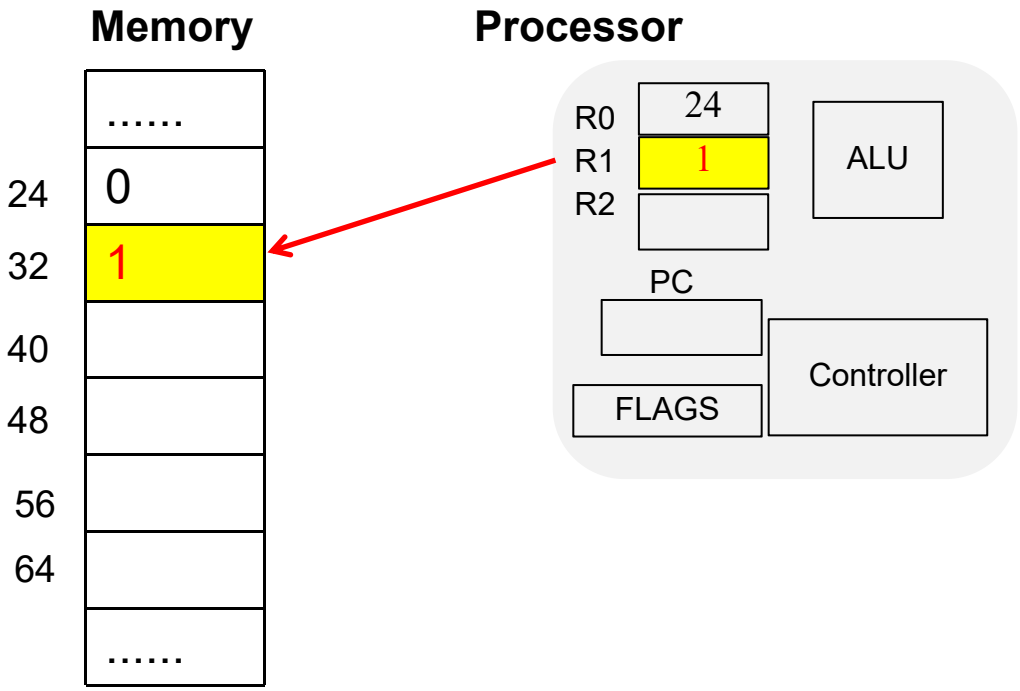

mem: Memory, 存储器
reg: Register，寄存器

# 指令2：MOV to Memory

MOV R1, M[R0+8]
将 R1 中的值存到
M[R0+8] 中

MOV reg, mem
将指定寄存器的值
存到指定内存地址

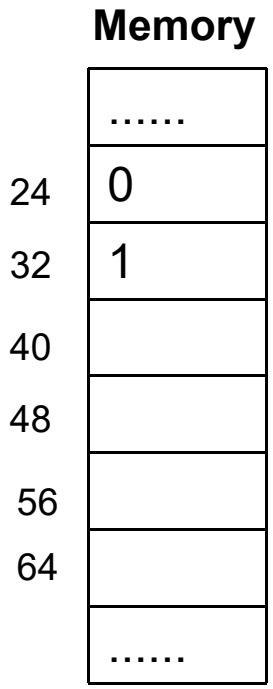
mem: Memory, 存储器
reg: Register，寄存器

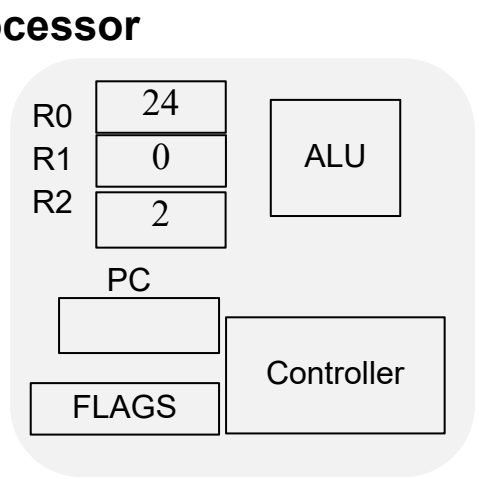


# 指令3：ADD

ADD M[R0+R2*8-16], R1
将 M[R0+R2*8-16]+R1的值
存到R1中

ADD mem, reg
计算mem与reg的和，
并存到reg中



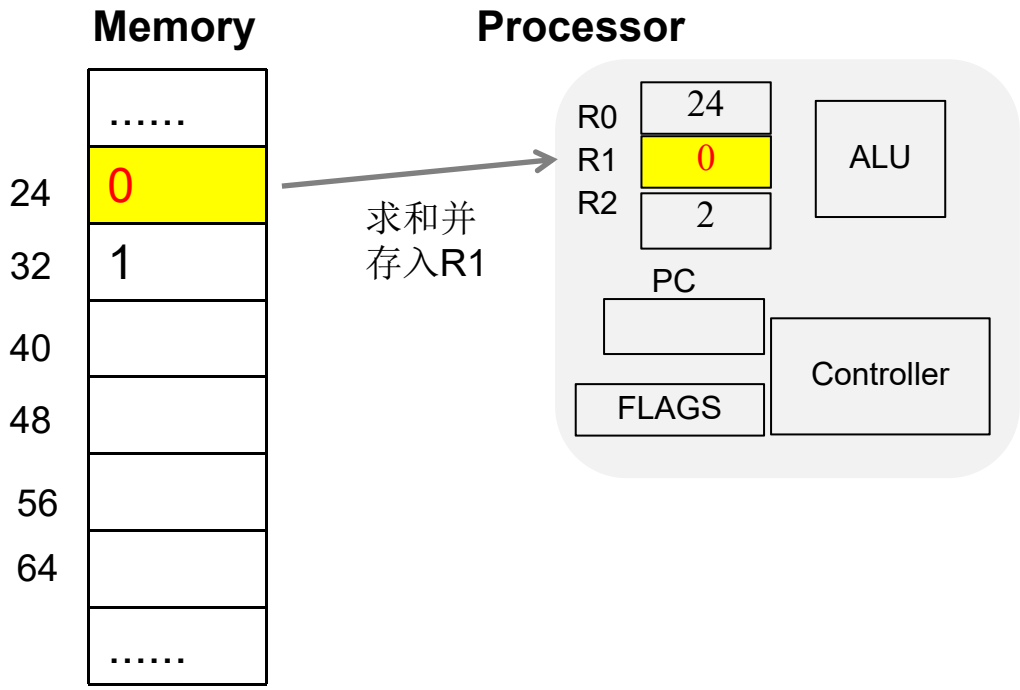




M[R0+R2*8-16] => M[24+2*8-16] => M[24]
M[24] + R1 => 0+0 => 0
将 0 存入 R1

# 指令3：ADD

ADD M[R0+R2*8-16], R1
将 M[R0+R2*8-16]+R1的值
存到R1中

ADD mem, reg
计算mem与reg的和，
并存到reg中

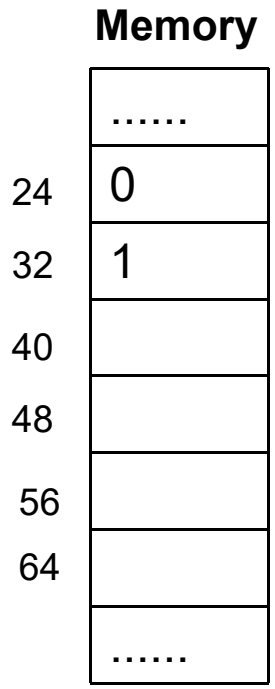


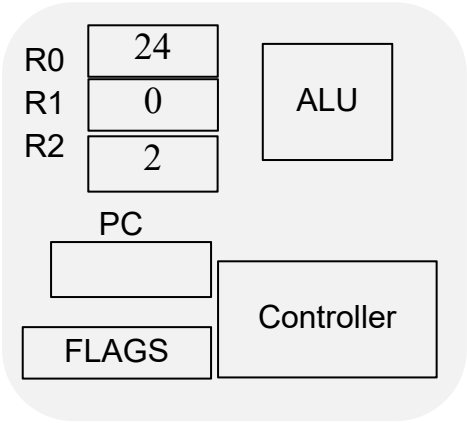

M[R0+R2*8-16] => M[24+2*8-16] => M[24]
M[24] + R1 => 0+0 => 0
将 0 存入 R1

# 指令4：Increment

INC R2
将 R2 中的值 加 1

INC reg
将寄存器中的值加1

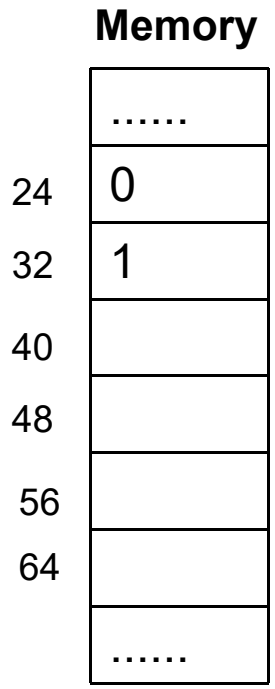


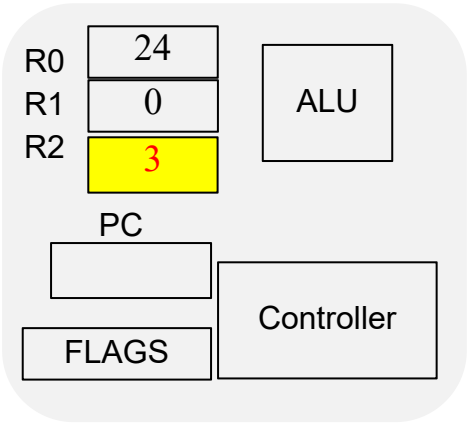



# 指令4：Increment

INC R2
将 R2 中的值 加 1

INC reg
将寄存器中的值加1

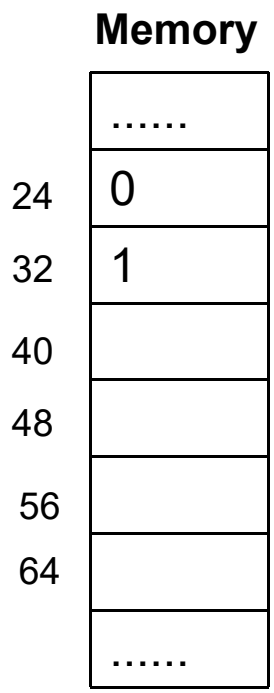


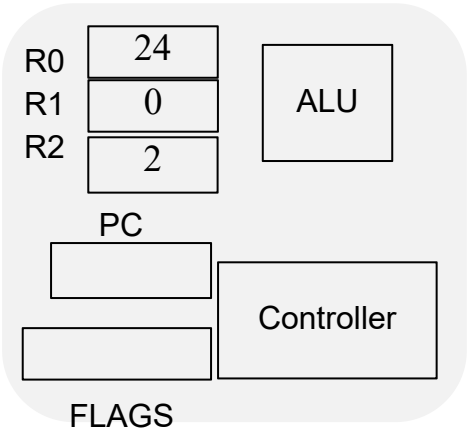



# 指令5：Compare

CMP 51, R2
比较 R2 < 51，在FLAGS寄存器中写入 ‘<’

CMP imm，reg
比较数字imm与reg中的值
如果reg < imm，则在FLAGS寄存器中写入 ‘<’；
如果reg ≥ imm，则清空FLAGS寄存器（不写值）

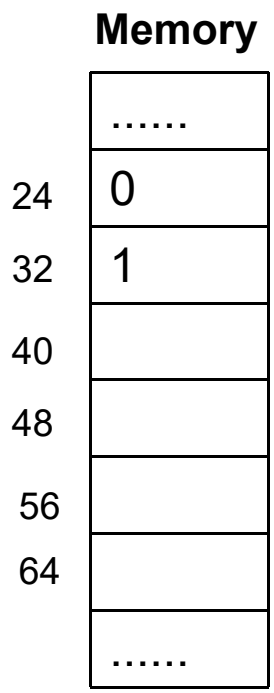


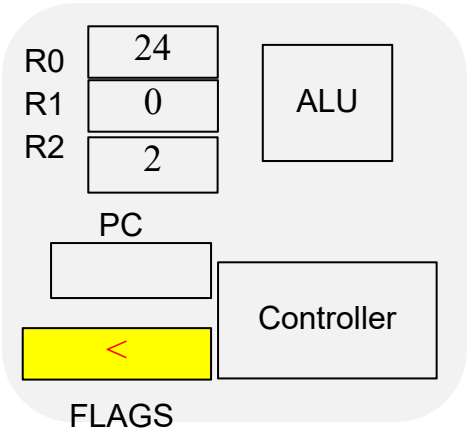



# 指令5：Compare

CMP 51, R2
比较 R2 < 51，在FLAGS寄存器中写入 ‘<’

CMP imm，reg
比较数字imm与reg中的值
如果reg < imm，则在FLAGS寄存器中写入 ‘<’；
如果reg ≥ imm，则清空FLAGS寄存器（不写值）

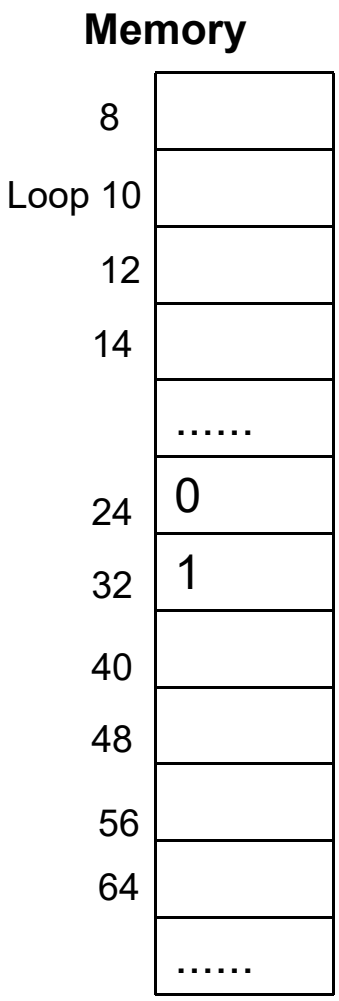


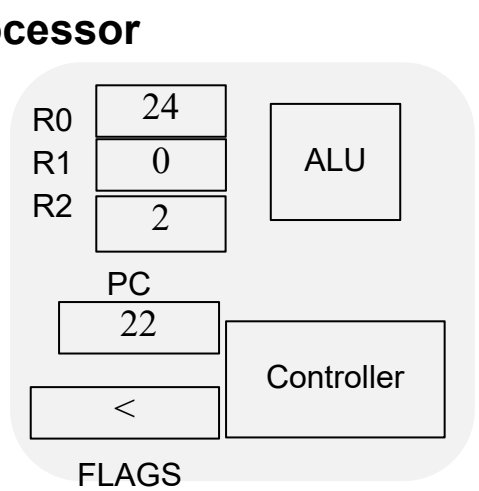




- 若执行指令CMP 2, R2　则FLAGS的状态是什么？
  FLAGS为空

# 指令6：Jump if Less than

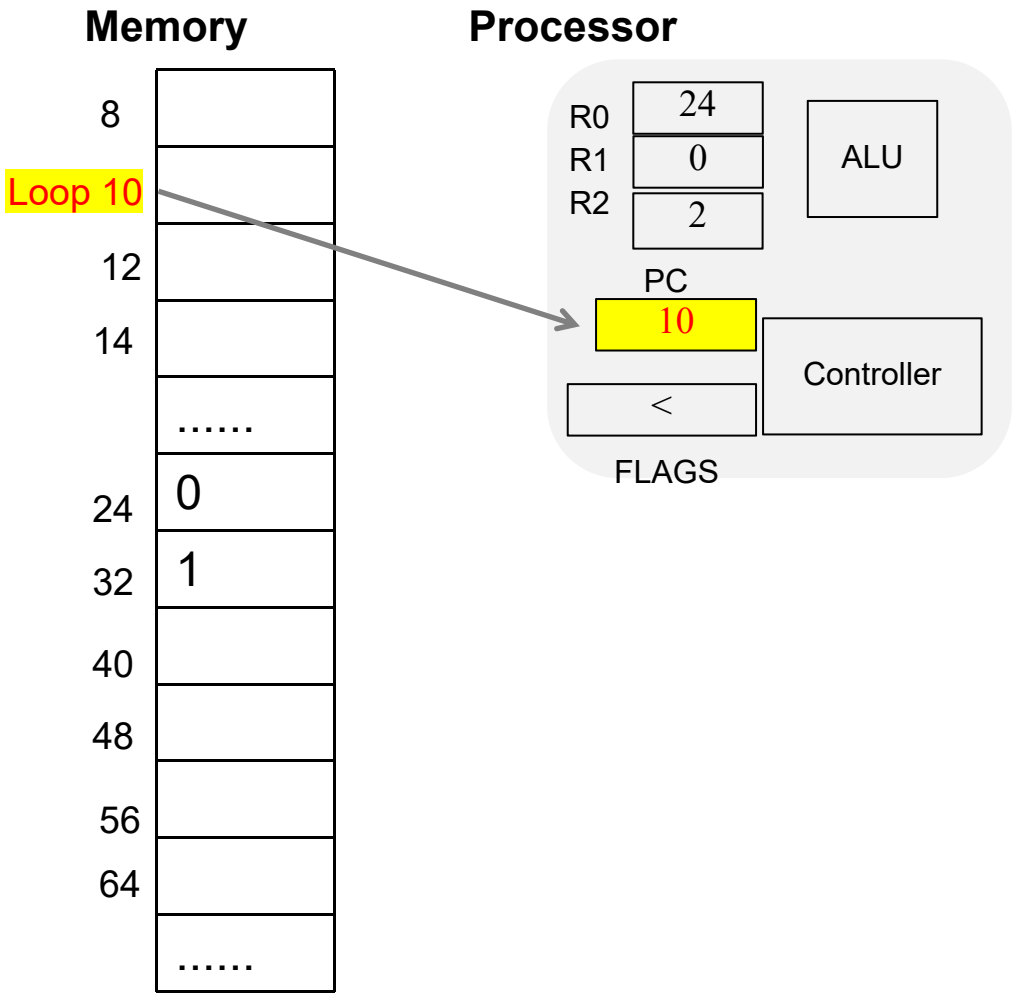
JL Loop
如果FLAGS中为 <，
则将PC修改为Loop标签对应的地址；
如果FLAGS为空，
则PC为当前指令的下一条指令



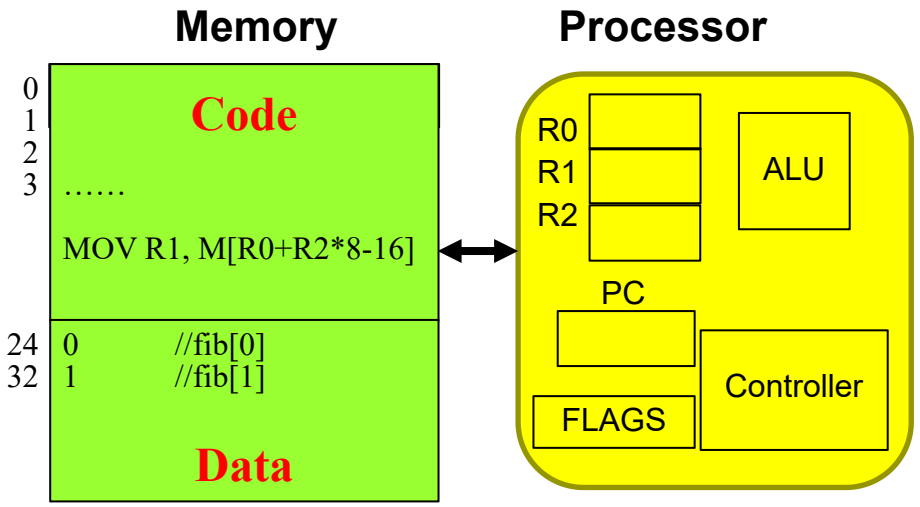
# 指令6：Jump if Less than

JL Loop
如果FLAGS中为 <，
则将PC修改为Loop标签对应的地址；
如果FLAGS为空，
则PC为当前指令的下一条指令



# Go代码 -> 汇编代码

- go代码如何与汇编代码对应
  - R0存放数组起始地址
  - R2存放循环索引 i
  - R1的作用 ？




在P8的六条指令中：
- 可以将**立即数**写入**寄存器**，
- 可以将**寄存器**的值写入**内存**，
- 不可以直接将**立即数**写入**内存**

```
// 计算fib[50]
fib[0] = 0
fib[1] = 1
for i := 2; i < 51; i++ {
    fib[i] = fib[i-1] + fib[i-2]
}
```

go代码片段

```
MOV 0, R1
MOV R1, M[R0]
MOV 1, R1
MOV R1, M[R0+8]
MOV 2, R2
MOV 0, R1    //Loop
ADD M[R0+R2*8-16], R1
ADD M[R0+R2*8-8], R1
MOV R1, M[R0+R2*8-0]
INC R2
CMP 51, R2
JL Loop
```

汇编代码片段

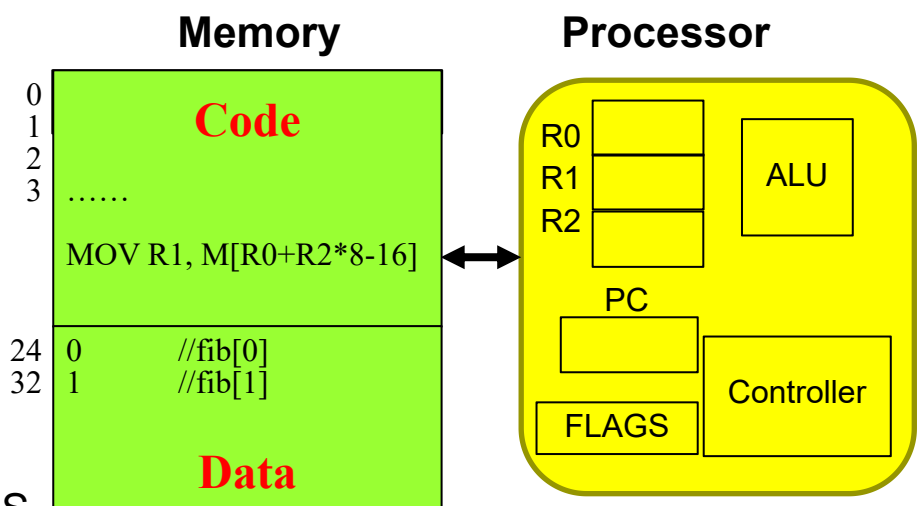
# 1.1 斐波那契计算机（FC）

人当编译器：Go代码 → 汇编代码

- 只执行所示Go代码
- 人工编译成汇编程序
  - 12条指令（每条指令2字节）
- 字节寻址存储器
  - 24字节存放代码
  - 408字节存放数据，即数组fib
- 寄存器
  - 三个64位通用寄存器R0、R1、R2，每个寄存器存放64位数
  - 程序计数器PC，存放下一条指令的地址
  - 状态寄存器FLAGS，存放指令执行的状态信息，如R2是否小于51
- 指令集（共有6条指令）
  - MOV to Register
  - MOV to Memory
  - ADD 加法指令
  - INC Increment 增1指令
  - CMP Compare 比较指令

    小于则置FLAGS为'<'，否则清空FLAGS
  - JL Jump if Less than 条件跳转

```
fib[0] = 0                  MOV 0, R1
                            MOV R1, M[R0]          //R0=24 initially
fib[1] = 1                  MOV 1, R1
                            MOV R1, M[R0+8]
for i := 2; i < 51; i++ {   MOV 2, R2              // i:=2
  fib[i] = fib[i-1] + fib[i-2]  MOV 0, R1          // label Loop
                            ADD M[R0+R2*8-16], R1
                            ADD M[R0+R2*8-8], R1
                            MOV R1, M[R0+R2*8-0]
                            INC R2                 // i++
                            CMP 51, R2             // i < 51?
}                           JL Loop          // Jump to Loop if Less than
```

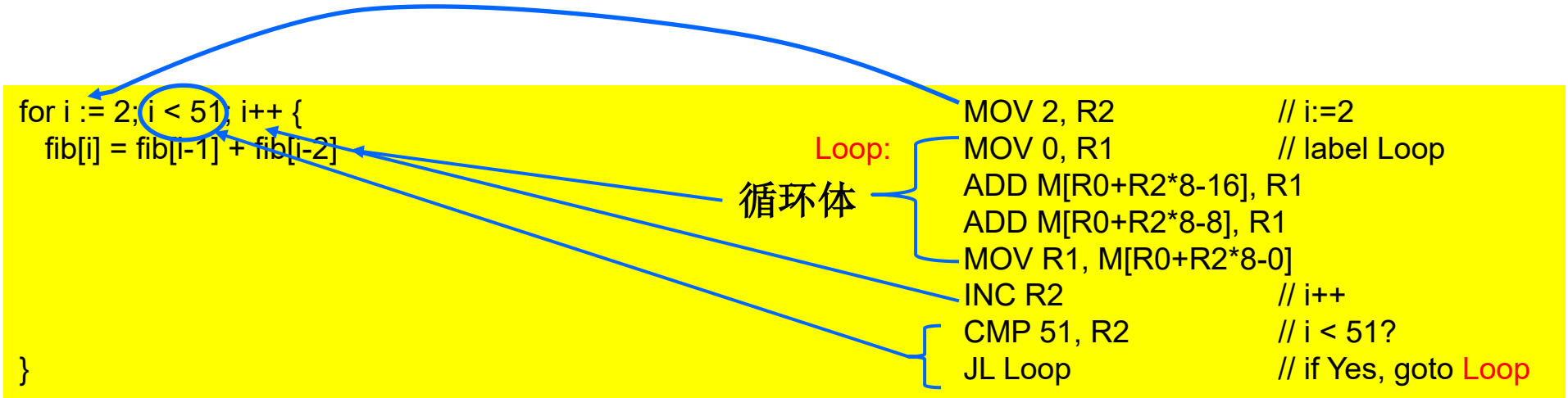


# FC初始状态 基址寄存器R0=24，为什么？

| 寄存器内容 | | 存储器内容 | | |
|---|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 | 注释 |
| FLAGS | | 0 | MOV 0, R1 | 0→R1；每条指令占2个地址 |
| PC | 0 | 2 | MOV R1, M[R0] | R1→M[R0] |
| R0 | 24 | 4 | MOV 1, R1 | 1→R1 |
| R1 | | 6 | MOV R1, M[R0+8] | R1→M[R0+8] |
| R2 | | 8 | MOV 2, R2 | 2→R2 |
| R0: 基址寄存器<br>初始值=24<br><br>R1: 累加器<br>R2: 索引寄存器<br><br>地址= 基址+索引*8+偏移量<br>Address=base+index*8+offset<br><br>fib[i-2]所在地址<br>=R0+R2*8 -16<br><br>fib[i-1]所在地址<br>=R0+R2*8 -8<br><br>fib[i]所在地址<br>=R0+R2*8 -0 | | 10 Loop | MOV 0, R1 | 0→R1；标签Loop=10 |
| | | 12 | ADD M[R0+R2*8-16], R1 | R1+ M[R0+R2*8-16] → R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 | R1+ M[R0+R2*8-8] → R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] | R1→ M[R0+R2*8-0] |
| | | 18 | INC R2 | R2+1→R2 |
| | | 20 | CMP 51, R2 | 如果R2<51，’<’→FLAGS |
| | | 22 | JL Loop | 如果FLAGS=’<’, Loop→PC |
| | | 24 | | fib[0]；每个数据占8个地址 |
| | | 32 | | fib[1] |
| | | 40 | | fib[2] |
| | | 48 | | fib[3] |
| | | …… | | …… |
| | | 424 | | fib[50] |

# 1.2 理解重点：多条指令如何支持循环与数组

- 注意
  - 汇编代码如何忠实反映了Go循环之变与不变
  - 数组与循环如何配合



# 理解重点：多条指令如何支持循环

- 以及数组
  - 在科学计算中，循环和数组往往配套出现
- 注意
  - 汇编代码如何忠实反映了循环之变与不变
  - 数组与循环如何配合

```
for i := 2; i < 51; i++ {
   fib[i] = fib[i-1] + fib[i-2]
```

0+fib[i-2]
0+fib[i-2]+fib[i-1]
fib[i]=fib[i-2]+fib[i-1]

循环体

```
                MOV 2, R2              // i:=2
Loop:           MOV 0, R1              // label Loop
                ADD M[R0+R2*8-16], R1
                ADD M[R0+R2*8-8], R1
                MOV R1, M[R0+R2*8-0]
                INC R2                 // i++
                CMP 51, R2             // i < 51?
                JL Loop                // if Yes, goto Loop
}
```

# 基址索引偏移量寻址模式 天然适配数组和循环

- **address = base + index*8 + offset**
  **实际地址 = 基址 + 索引*比例因子 + 偏移量**
- 进入for循环，i:=2
  - 基址寄存器R0=24
  - 索引寄存器R2=2
  - 比例因子=8，因为fib[i]是64位整数
  - 赋值语句fib[i] = fib[i-1] + fib[i-2]编译成

```
MOV 0, R1          // 累加器初始化为0
ADD M[R0+R2*8-16], R1
ADD M[R0+R2*8-8], R1
MOV R1, M[R0+R2*8-0]
```

  - 第一条加法指令实现0+fib[0]

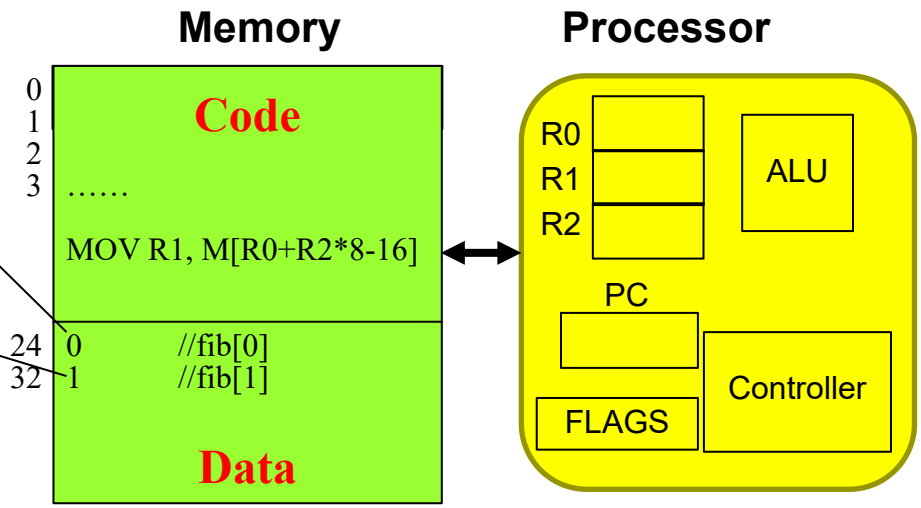
    R1+ M[R0+R2*8-16] → R1，即

    0 + M[24+2*8-16] → R1，即0+fib[0] → R1
  - 第二条加法指令实现0+fib[0]+fib[1]

    R1+ M[R0+R2*8-8] → R1，即

    0 + M[24+2*8-8] → R1，即0+fib[1] → R1

```
fib[0] = 0                          MOV 0, R1
                                    MOV R1, M[R0] //R0=12 initially
fib[1] = 1                          MOV 1, R1
                                    MOV R1, M[R0+8]
for i := 2; i < 51; i++ {           MOV 2, R2    // i:=2
   fib[i] = fib[i-1] + fib[i-2]     MOV 0, R1    // label Loop
                                    ADD M[R0+R2*8-16], R1
                                    ADD M[R0+R2*8-8], R1
                                    MOV R1, M[R0+R2*8-0]
                                    INC R2          // i++
                                    CMP 51, R2  // i < 51?
}                                   JL Loop         // if Yes, goto Loop
```



# 基址索引偏移量寻址模式 天然适配数组和循环

- **address = base + index*8 + offset**
  **实际地址 = 基址 + 索引*比例因子 + 偏移量**
- 进入for循环，i:=2
  - 基址寄存器R0=24
  - 索引寄存器R2=2
  - 比例因子=8，因为fib[i]是64位整数
  - 赋值语句fib[i] = fib[i-1] + fib[i-2]编译成

    MOV 0, R1

    ADD M[R0+R2*8-16], R1

    ADD M[R0+R2*8-8], R1

    MOV R1, M[R0+R2*8-0]
  - 第一条加法指令实现0+fib[0]

    R1+ M[R0+R2*8-16] → R1，即

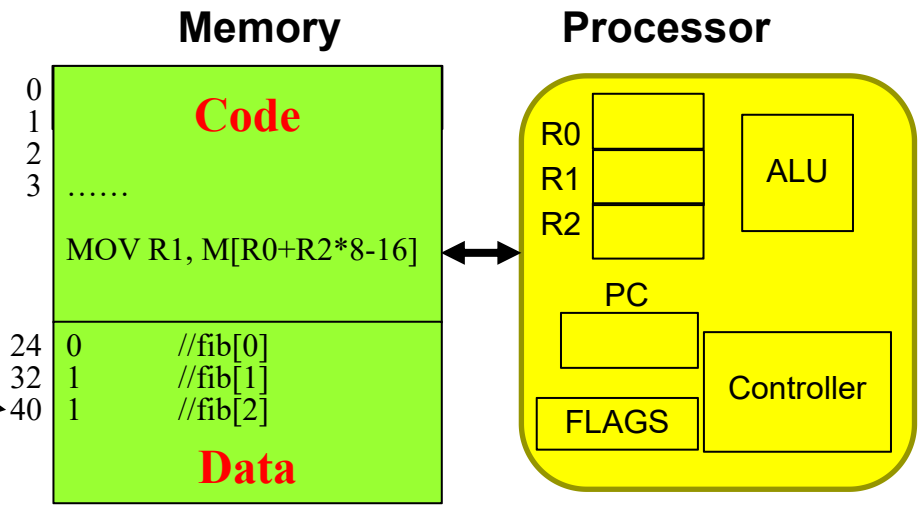
    0 + M[24+2*8-16] → R1，即0+fib[0] → R1
  - 第二条加法指令实现0+fib[0]+fib[1]

    R1+ M[R0+R2*8-8] → R1，即

    0 + M[24+2*8-8] → R1，即0+fib[1] → R1
  - 指令MOV实现fib[2]=0+fib[0]+fib[1]

R1 → M[24+2*8-0] ，即1→M[40]，即1→fib[2]

```
fib[0] = 0                          MOV 0, R1
                                    MOV R1, M[R0] //R0=12 initially
fib[1] = 1                          MOV 1, R1
                                    MOV R1, M[R0+8]
for i := 2; i < 51; i++ {           MOV 2, R2    // i:=2
   fib[i] = fib[i-1] + fib[i-2]     MOV 0, R1    // label Loop
                                    ADD M[R0+R2*8-16], R1
                                    ADD M[R0+R2*8-8], R1
                                    MOV R1, M[R0+R2*8-0]
                                    INC R2       // i++
                                    CMP 51, R2   // i < 51?
}                                   JL Loop      // if Yes, goto Loop
```



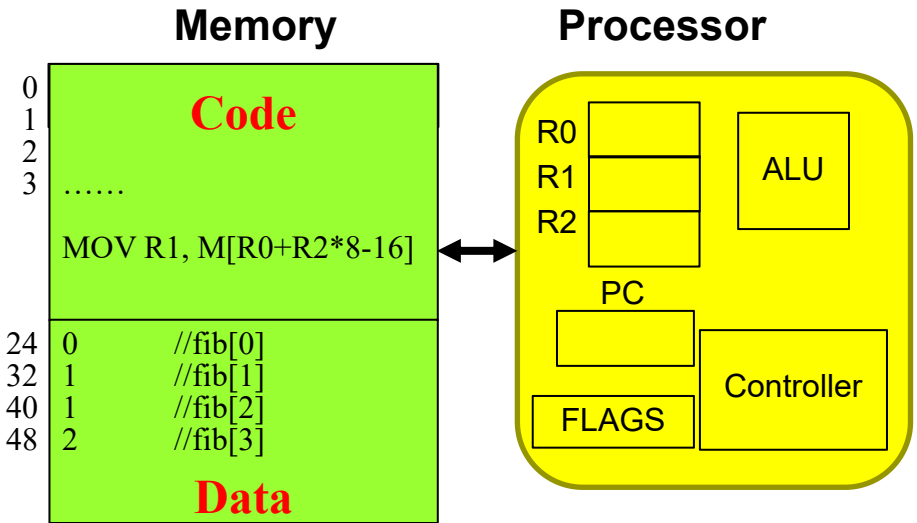
# 基址索引偏移量寻址模式　天然适配数组和循环

- **address = base + index*8 + offset**
  实际地址 = 基址 + 索引*比例因子 + 偏移量
- 后面三条指令后，
  进入下一次迭代，i:=3
  - 基址寄存器R0=24
  - 索引寄存器R2=**3**（变了！）
  - 比例因子=8，因为fib[i]是64位整数
  - 第一条加法指令实现0+fib[1]
    R1+ M[R0+R2*8-16] → R1，即
    0 + M[24+**3***8-16] → R1，即0+M[32] → R1
    即0+fib[1] → R1，即0+1 → R1
  - 第二条加法指令实现0+fib[1]+fib[2]
    R1+ M[R0+R2*8-8] → R1，即
    1 + M[24+**3***8-8] → R1，即1+M[40] → R1
    即1+fib[2] → R1，即1+1 → R1
  - 指令MOV实现fib[3]=0+fib[1]+fib[2]

R1 → M[24+**3***8-0]，即2→M[48]，即2→fib[3]

**Loop中的7条指令不变**
**唯一变了的是R2的值**

```
fib[0] = 0                          MOV 0, R1
                                    MOV R1, M[R0] //R0=12 initially
fib[1] = 1                          MOV 1, R1
                                    MOV R1, M[R0+8]
for i := 2; i < 51; i++ {           MOV 2, R2    // i:=2
   fib[i] = fib[i-1] + fib[i-2]     MOV 0, R1    // label Loop
                                    ADD M[R0+R2*8-16], R1
                                    ADD M[R0+R2*8-8], R1
                                    MOV R1, M[R0+R2*8-0]
                                    INC R2       // i++
                                    CMP 51, R2   // i < 51?
}                                   JL Loop      // if Yes, goto Loop
```



# 1.3 逐步验证 A step-by-step walkthrough

- 理解“计算机如何支持循环与数组”
- 验证斐波那契计算机满足冯诺依曼机五要点
  - 二进制表示
    - 满足，不过人在逐步验证过程中采用熟悉的十进制
  - P-M-I/O
    - 满足，不过忽略了I/O子系统
  - 存储程序计算机
    - 满足，程序存放在内存地址0~23，数据存放在地址24~431
  - 指令驱动
    - 满足，可在逐步验证中确认
  - 串行执行
    - 满足，可在逐步验证中确认
    - 每一步有两处改变：PC与内存单元（寄存器可看成是特殊的内存单元）

# 第一步 Step 1: MOV 0, 1

访问https://course.things.ac.cn:10088/exp/basic_programming 第四次提交

操作步骤：
1. 填写当前步骤
2. 点击 **插入1步**
3. 切换到新增的 **step**
4. 点击 **复制上一步状态**
5. ... ...

| 处理器内容 | | 存储器内容 | |
|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 |
| FLAGS | | 0 | MOV 0, R1 |
| PC | 0 | 2 | MOV R1, M[R0] |
| R0 | 24 | 4 | MOV 1, R1 |
| R1 | | 6 | MOV R1, M[R0+8] |
| R2 | | 8 | MOV 2, R2 |
| | | 10 Loop | MOV 0, R1 |
| | | 12 | ADD M[R0+R2*8-16], R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] |
| | | 18 | INC R2 |
| | | 20 | CMP 51, R2 |
| | | 22 | JL Loop |
| | | 24 | //fib[0] |
| | | 32 | //fib[1] |
| | | 40 | //fib[2] |

初始状态

| 处理器内容 | | 存储器内容 | |
|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 |
| FLAGS | | 0 | MOV 0, R1 |
| PC | 2 | 2 | MOV R1, M[R0] |
| R0 | 24 | 4 | MOV 1, R1 |
| R1 | 0 | 6 | MOV R1, M[R0+8] |
| R2 | | 8 | MOV 2, R2 |
| | | 10 Loop | MOV 0, R1 |
| | | 12 | ADD M[R0+R2*8-16], R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] |
| | | 18 | INC R2 |
| | | 20 | CMP 51, R2 |
| | | 22 | JL Loop |
| | | 24 | //fib[0] |
| | | 32 | //fib[1] |
| | | 40 | //fib[2] |

第1步之后状态

| 处理器内容 | | 存储器内容 | |
|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 |
| FLAGS | | 0 | MOV 0, R1 |
| PC | 2 | 2 | MOV R1, M[R0] |
| R0 | 24 | 4 | MOV 1, R1 |
| R1 | 0 | 6 | MOV R1, M[R0+8] |
| R2 | | 8 | MOV 2, R2 |
| | | 10 Loop | MOV 0, R1 |
| | | 12 | ADD M[R0+R2*8-16], R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] |
| | | 18 | INC R2 |
| | | 20 | CMP 51, R2 |
| | | 22 | JL Loop |
| | | 24 | //fib[0] |
| | | 32 | //fib[1] |
| | | 40 | //fib[2] |

Step 1

| 处理器内容 | | 存储器内容 | |
|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 |
| FLAGS | | 0 | MOV 0, R1 |
| PC | 4 | 2 | MOV R1, M[R0] |
| R0 | 24 | 4 | MOV 1, R1 |
| R1 | 0 | 6 | MOV R1, M[R0+8] |
| R2 | | 8 | MOV 2, R2 |
| | | 10 Loop | MOV 0, R1 |
| | | 12 | ADD M[R0+R2*8-16], R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] |
| | | 18 | INC R2 |
| | | 20 | CMP 51, R2 |
| | | 22 | JL Loop |
| | | 24 | 0 |
| | | 32 | |
| | | 40 | |

Step 2

| 处理器内容 | | 存储器内容 | |
|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 |
| FLAGS | | 0 | MOV 0, R1 |
| PC | 6 | 2 | MOV R1, M[R0] |
| R0 | 24 | 4 | MOV 1, R1 |
| R1 | 1 | 6 | MOV R1, M[R0+8] |
| R2 | | 8 | MOV 2, R2 |
| | | 10 Loop | MOV 0, R1 |
| | | 12 | ADD M[R0+R2*8-16], R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] |
| | | 18 | INC R2 |
| | | 20 | CMP 51, R2 |
| | | 22 | JL Loop |
| | | 24 | 0 //fib[0] |
| | | 32 | //fib[1] |
| | | 40 | //fib[2] |

Step 3

| 处理器内容 | | 存储器内容 | |
|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 |
| FLAGS | | 0 | MOV 0, R1 |
| PC | 8 | 2 | MOV R1, M[R0] |
| R0 | 24 | 4 | MOV 1, R1 |
| R1 | 1 | 6 | MOV R1, M[R0+8] |
| R2 | | 8 | MOV 2, R2 |
| | | 10 Loop | MOV 0, R1 |
| | | 12 | ADD M[R0+R2*8-16], R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] |
| | | 18 | INC R2 |
| | | 20 | CMP 51, R2 |
| | | 22 | JL Loop |
| | | 24 | 0 //fib[0] |
| | | 32 | 1 //fib[1] |
| | | 40 | //fib[2] |

Step 4

| 处理器内容 | | 存储器内容 | |
|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 |
| FLAGS | | 0 | MOV 0, R1 |
| PC | 10 | 2 | MOV R1, M[R0] |
| R0 | 24 | 4 | MOV 1, R1 |
| R1 | 1 | 6 | MOV R1, M[R0+8] |
| R2 | 2 | 8 | MOV 2, R2 |
| | | 10 Loop | MOV 0, R1 |
| | | 12 | ADD M[R0+R2*8-16], R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] |
| | | 18 | INC R2 |
| | | 20 | CMP 51, R2 |
| | | 22 | JL Loop |
| | | 24 | 0 //fib[0] |
| | | 32 | 1 //fib[1] |
| | | 40 | //fib[2] |

Step 5

| 处理器内容 | | 存储器内容 | |
|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 |
| FLAGS | | 0 | MOV 0, R1 |
| PC | 12 | 2 | MOV R1, M[R0] |
| R0 | 24 | 4 | MOV 1, R1 |
| R1 | 0 | 6 | MOV R1, M[R0+8] |
| R2 | 2 | 8 | MOV 2, R2 |
| | | 10 Loop | MOV 0, R1 |
| | | 12 | ADD M[R0+R2*8-16], R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] |
| | | 18 | INC R2 |
| | | 20 | CMP 51, R2 |
| | | 22 | JL Loop |
| | | 24 | 0 //fib[0] |
| | | 32 | 1 //fib[1] |
| | | 40 | //fib[2] |

Step 6

Step 7

| 处理器内容 | | 存储器内容 | |
|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 |
| FLAGS | | 0 | MOV 0, R1 |
| PC | 14 | 2 | MOV R1, M[R0] |
| R0 | 24 | 4 | MOV 1, R1 |
| R1 | 0 | 6 | MOV R1, M[R0+8] |
| R2 | 2 | 8 | MOV 2, R2 |
| | | 10 Loop | MOV 0, R1 |
| | | 12 | ADD M[R0+R2*8-16], R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] |
| | | 18 | INC R2 |
| | | 20 | CMP 51, R2 |
| | | 22 | JL Loop |
| | | 24 | 0 //fib[0] |
| | | 32 | 1 //fib[1] |
| | | 40 | //fib[2] |

Step 8

| 处理器内容 | | 存储器内容 | |
|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 |
| FLAGS | | 0 | MOV 0, R1 |
| PC | 16 | 2 | MOV R1, M[R0] |
| R0 | 24 | 4 | MOV 1, R1 |
| R1 | 1 | 6 | MOV R1, M[R0+8] |
| R2 | 2 | 8 | MOV 2, R2 |
| | | 10 Loop | MOV 0, R1 |
| | | 12 | ADD M[R0+R2*8-16], R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] |
| | | 18 | INC R2 |
| | | 20 | CMP 51, R2 |
| | | 22 | JL Loop |
| | | 24 | 0 //fib[0] |
| | | 32 | 1 //fib[1] |
| | | 40 | //fib[2] |

| 处理器内容 | | 存储器内容 | |
|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 |
| FLAGS | | 0 | MOV 0, R1 |
| PC | 18 | 2 | MOV R1, M[R0] |
| R0 | 24 | 4 | MOV 1, R1 |
| R1 | 1 | 6 | MOV R1, M[R0+8] |
| R2 | 2 | 8 | MOV 2, R2 |
| | | 10 Loop | MOV 0, R1 |
| | | 12 | ADD M[R0+R2*8-16], R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] |
| | | 18 | INC R2 |
| | | 20 | CMP 51, R2 |
| | | 22 | JL Loop |
| | | 24 | 0 //fib[0] |
| | | 32 | 1 //fib[1] |
| | | 40 | 1 //fib[2] |

**Step 9**

| 处理器内容 | | 存储器内容 | |
|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 |
| FLAGS | | 0 | MOV 0, R1 |
| PC | 20 | 2 | MOV R1, M[R0] |
| R0 | 24 | 4 | MOV 1, R1 |
| R1 | 1 | 6 | MOV R1, M[R0+8] |
| R2 | 3 | 8 | MOV 2, R2 |
| | | 10 Loop | MOV 0, R1 |
| | | 12 | ADD M[R0+R2*8-16], R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] |
| | | 18 | INC R2 |
| | | 20 | CMP 51, R2 |
| | | 22 | JL Loop |
| | | 24 | 0 //fib[0] |
| | | 32 | 1 //fib[1] |
| | | 40 | 1 //fib[2] |

**Step 10**

| 处理器内容 | | 存储器内容 | |
|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 |
| FLAGS | < | 0 | MOV 0, R1 |
| PC | 22 | 2 | MOV R1, M[R0] |
| R0 | 24 | 4 | MOV 1, R1 |
| R1 | 1 | 6 | MOV R1, M[R0+8] |
| R2 | 3 | 8 | MOV 2, R2 |
| | | 10 Loop | MOV 0, R1 |
| | | 12 | ADD M[R0+R2*8-16], R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] |
| | | 18 | INC R2 |
| | | 20 | CMP 51, R2 |
| | | 22 | JL Loop |
| | | 24 | 0 //fib[0] |
| | | 32 | 1 //fib[1] |
| | | 40 | 1 //fib[2] |

**Step 11**

| 处理器内容 | | 存储器内容 | |
|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 |
| FLAGS | < | 0 | MOV 0, R1 |
| PC | 10 | 2 | MOV R1, M[R0] |
| R0 | 24 | 4 | MOV 1, R1 |
| R1 | 1 | 6 | MOV R1, M[R0+8] |
| R2 | 3 | 8 | MOV 2, R2 |
| | | 10 Loop | MOV 0, R1 |
| | | 12 | ADD M[R0+R2*8-16], R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] |
| | | 18 | INC R2 |
| | | 20 | CMP 51, R2 |
| | | 22 | JL Loop |
| | | 24 | 0 //fib[0] |
| | | 32 | 1 //fib[1] |
| | | 40 | 1 //fib[2] |

**Step 12**

# 现场练习
# Step 13: MOV 0, R1

| 处理器内容 | | 存储器内容 | |
|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 |
| FLAGS | ? | 0 | MOV 0, R1 |
| PC | ? | 2 | MOV R1, M[R0] |
| R0 | ? | 4 | MOV 1, R1 |
| R1 | ? | 6 | MOV R1, M[R0+8] |
| R2 | ? | 8 | MOV 2, R2 |
| | | 10 Loop | MOV 0, R1 |
| | | 12 | ADD M[R0+R2*8-16], R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] |
| | | 18 | INC R2 |
| | | 20 | CMP 51, R2 |
| | | 22 | JL Loop |
| | | 24 | ? //fib[0] |
| | | 32 | ? //fib[1] |
| | | 40 | ? //fib[2] |
| | | ? | ? //fib[3] |

https://course.things.ac.cn:10088/exp/basic_programming 选择“第四次提交”

Step 14
Step 15
Step 16
Step 17
Step 18
Step 19

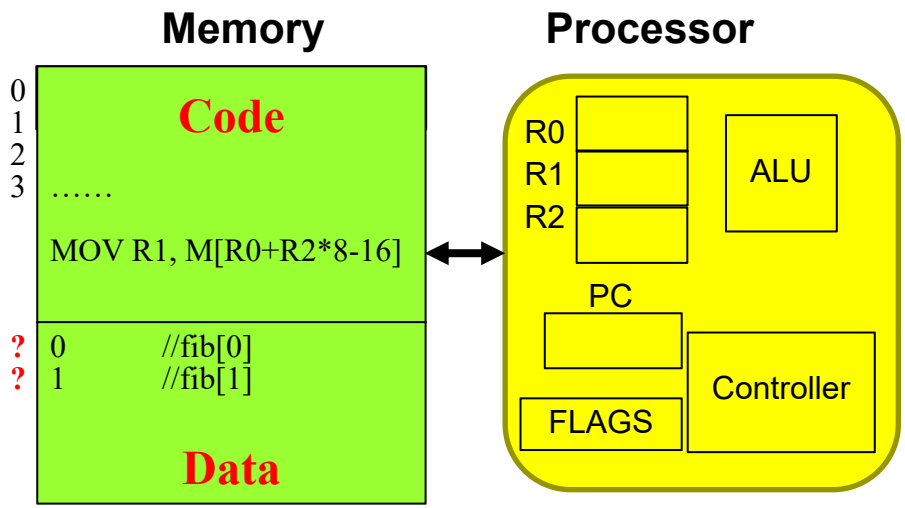
# FC缺点：当R2==51，指令JL不会执行Loop→PC操作；
## 不转移到Loop，而是执行PC+2→PC操作，执行顺序下一条指令（在地址24）

| 寄存器内容 | | 存储器内容 | | |
|---|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 | 注释 |
| FLAGS | | 0 | MOV 0, R1 | 0→R1；每条指令占2个地址 |
| PC | 0 | 2 | MOV R1, M[R0] | R1→M[R0] |
| R0 | 24 | 4 | MOV 1, R1 | 1→R1 |
| R1 | | 6 | MOV R1, M[R0+8] | R1→M[R0+8] |
| R2 | | 8 | MOV 2, R2 | 2→R2 |
| R0: 基址寄存器<br>初始值=24<br><br>R1: 累加器<br>R2: 索引寄存器<br><br>地址= 基址+索引*8+偏移量<br>Address=base+index*8+offset<br><br>fib[i-2]所在地址<br>=R0+R2*8 -16<br><br>fib[i-1]所在地址<br>=R0+R2*8 -8<br><br>fib[i]所在地址<br>=R0+R2*8 -0 | | 10 Loop | MOV 0, R1 | 0→R1；标签Loop=10 |
| | | 12 | ADD M[R0+R2*8-16], R1 | R1+ M[R0+R2*8-16] → R1 |
| | | 14 | ADD M[R0+R2*8-8], R1 | R1+ M[R0+R2*8-8] → R1 |
| | | 16 | MOV R1, M[R0+R2*8-0] | R1→ M[R0+R2*8-0] |
| | | 18 | INC R2 | R2+1→R2 |
| | | 20 | CMP 51, R2 | 如果R2<51，’<’→FLAGS |
| | | 22 | JL Loop | 如果FLAGS=’<’, Loop→PC |
| | | 24 | | fib[0]；每个数据占8个地址 |
| | | 32 | | fib[1] |
| | | 40 | | fib[2] |
| | | 48 | | fib[3] |
| | | …… | | …… |
| | | 424 | | fib[50] |

# 2. 改进型斐波那契计算机，计算F(3)
# Refined Fibonacci Computer（RFC）

- 人工编译成汇编程序
  - 13条指令
- 64位计算机
  - 64比特指令与数据
    - 104字节存放代码
    - 408字节存放数据，即数组fib
- 寄存器
  - 不变
- 指令集（共有7条指令）
  - MOV to Register
  - MOV to Memory
  - ADD 加法指令
  - INC Increment 增1指令
  - CMP Compare 比较指令
  - JL Jump if Less than 条件跳转
  - HALT 停机指令

```
fib[0] = 0                  MOV 0, R1
                            MOV R1, M[R0]          //R0=? initially
fib[1] = 1                  MOV 1, R1
                            MOV R1, M[R0+8]
for i := 2; i < 4; i++ {    MOV 2, R2              // i:=2
   fib[i] = fib[i-1] + fib[i-2]
                            MOV 0, R1              // label Loop
                            ADD M[R0+R2*8-16], R1
                            ADD M[R0+R2*8-8], R1
                            MOV R1, M[R0+R2*8-0]
                            INC R2                 // i++
                            CMP 51, R2             // i < 51?
}                           JL Loop          // Jump to Loop if Less than
                            HALT                   // 停机
```



# RFC初始状态

| 寄存器内容 | | 存储器内容 | | |
|---|---|---|---|---|
| 寄存器 | 值 | 地址 | 指令 | 注释 |
| FLAGS | | 0 | MOV 0, R1 | 0→R1；每条指令占8个字节地址 |
| PC | 0 | ? | MOV R1, M[R0] | R1→M[R0] |
| R0 | ? | ? | MOV 1, R1 | 1→R1 |
| R1 | | ? | MOV R1, M[R0+8] | R1→M[R0+8] |
| R2 | | ? | MOV 2, R2 | 2→R2 |
| R0: 基址寄存器<br>初始值=?<br><br>R1: 累加器<br>R2: 索引寄存器<br><br>地址= 基址+索引*8+偏移量<br>Address=base+index*8+offset<br><br>fib[i-2]所在地址<br>=R0+R2*8 -16<br><br>fib[i-1]所在地址<br>=R0+R2*8 -8<br><br>fib[i]所在地址<br>=R0+R2*8 -0 | | ? | MOV 0, R1 // 标签为Loop | 0→R1；标签Loop=? |
| | | ? | ADD M[R0+R2*8-16], R1 | R1+ M[R0+R2*8-16] → R1 |
| | | ? | ADD M[R0+R2*8-8], R1 | R1+ M[R0+R2*8-8] → R1 |
| | | ? | MOV R1, M[R0+R2*8-0] | R1→ M[R0+R2*8-0] |
| | | ? | INC R2 | R2+1→R2 |
| | | ? | CMP 4, R2 | 如果R2<4，'<'→FLAGS |
| | | ? | JL Loop | 如果FLAGS='<', Loop→PC |
| | | ? | HALT | 停机 |
| | | ? | | fib[0] 每个数据占8个字节地址 |
| | | ? | | fib[1] |
| | | ? | | fib[2] |
| | | ? | | fib[3] |

# 现场问题：RFC执行多少步停机？

- ？
- 假设是P步。
- 现场填空：初始状态和Step 1~P
- 确定后提交RFC

# 现场问题：RFC执行多少步停机？

- P = 20。
- 现场填空：初始状态和Step 1~20
- 确定后提交RFC

提交网址： https://course.things.ac.cn:10088/exp/basic_programming
选择“第五次提交”
截止时间：2025/3/30 23:30