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

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

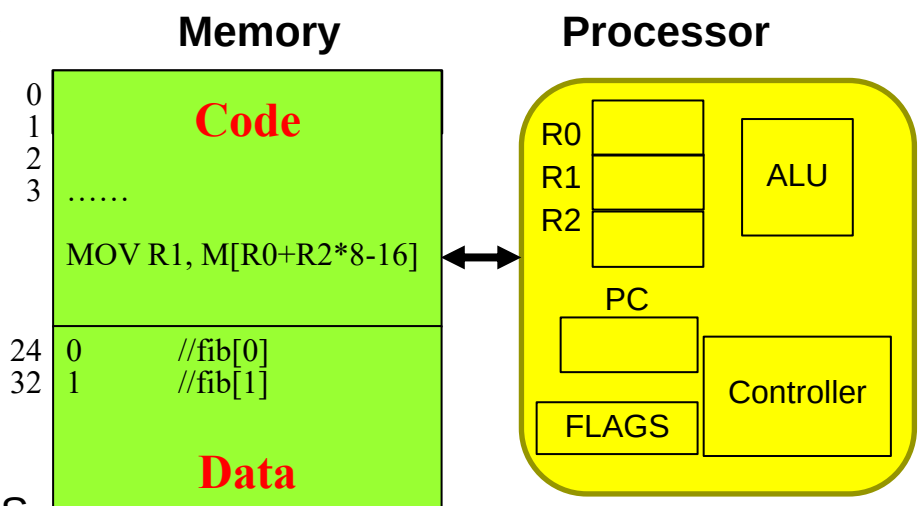
1.1 斐波那契计算机（FC）

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
- 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=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          // Jump to Loop if Less than
```

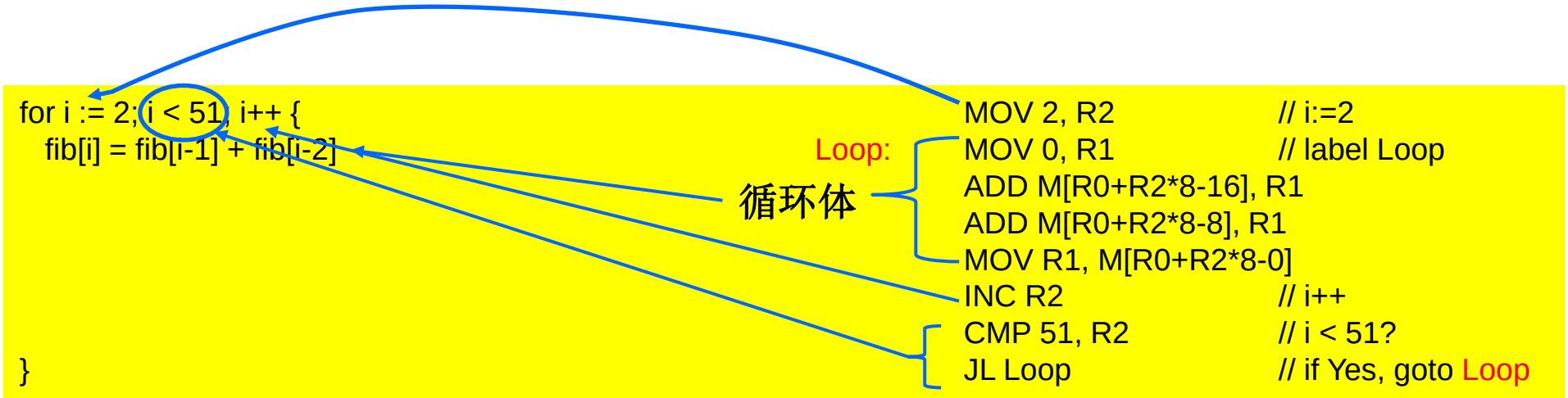


FC初始状态 基址寄存器R0=24，为什么？

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 初始值=24 R1: 累加器 R2: 索引寄存器 地址= 基址+索引*8+偏移量 Address=base+index*8+offset fib[i-2]所在地址 =R0+R2*8 -16 fib[i-1]所在地址 =R0+R2*8 -8 fib[i]所在地址 =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]
```

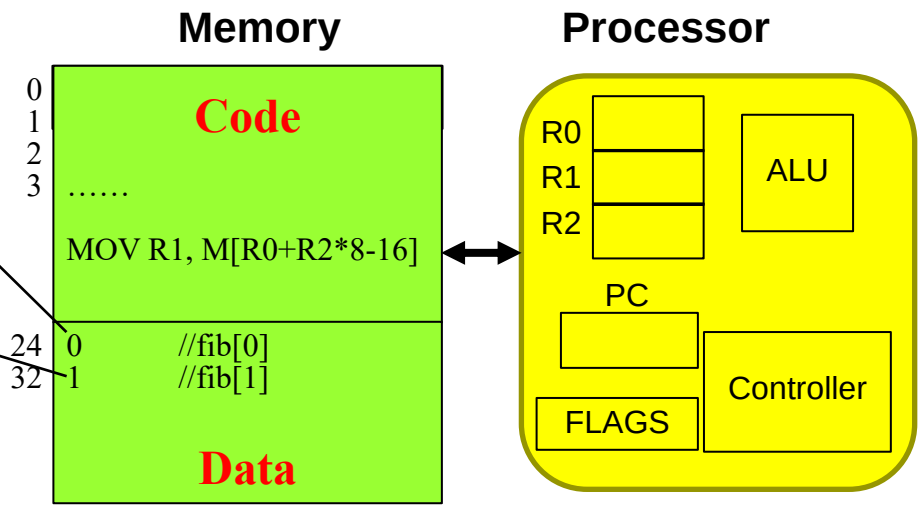
  - 第一条加法指令实现0+fib[0]

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

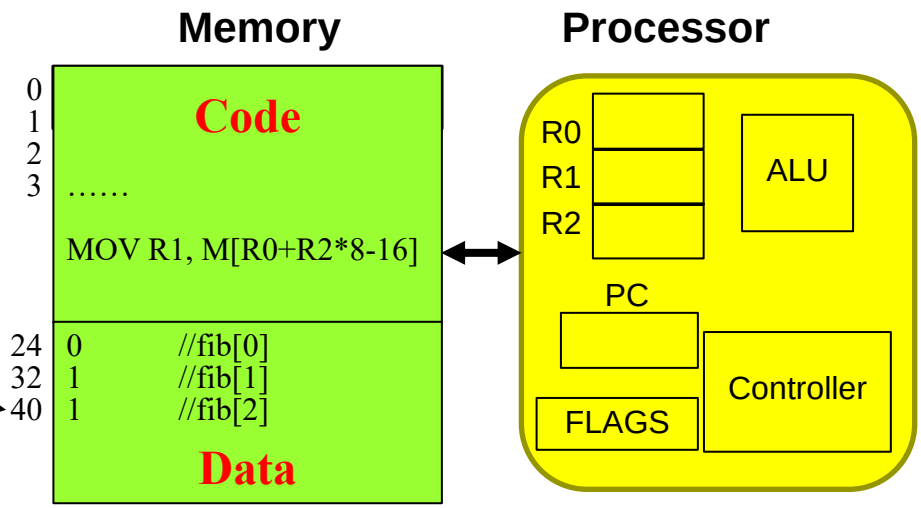
    0 + M[24+2*8-16] → R1，即0+fib[0] → R1
  - 第二条加法指令实现0+fib[0]+fib[1]

    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

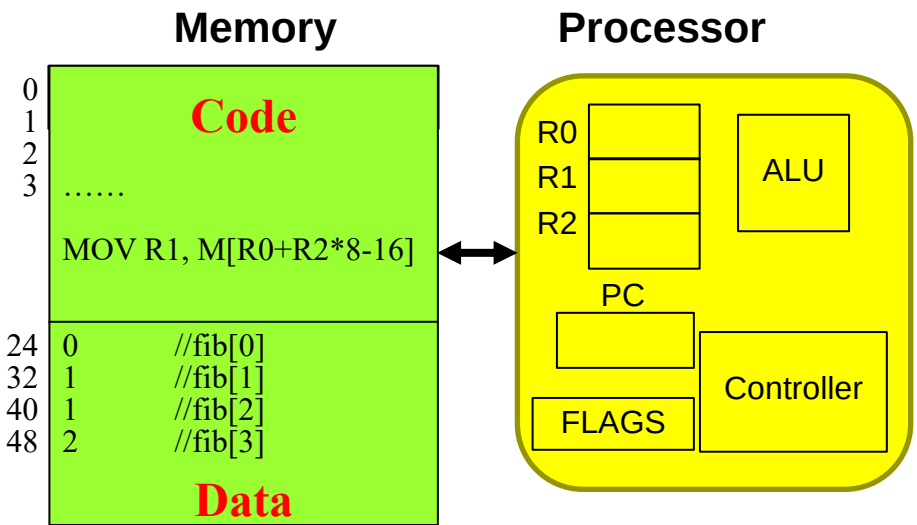
- **address = base + index*8 + offset**
  实际地址 = 基址 + 索引*比例因子 + 偏移量
- 后面三条指令后，
  进入下一次迭代，i:=3
  - 基址寄存器R0=24
  - 索引寄存器R2=**3**（变了！）
  - 比例因子=8，因为fib[i]是64位整数

  - 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

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

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	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 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	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 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	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 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	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]

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://www.solid.things.ac.cn:7245/basic_programming

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

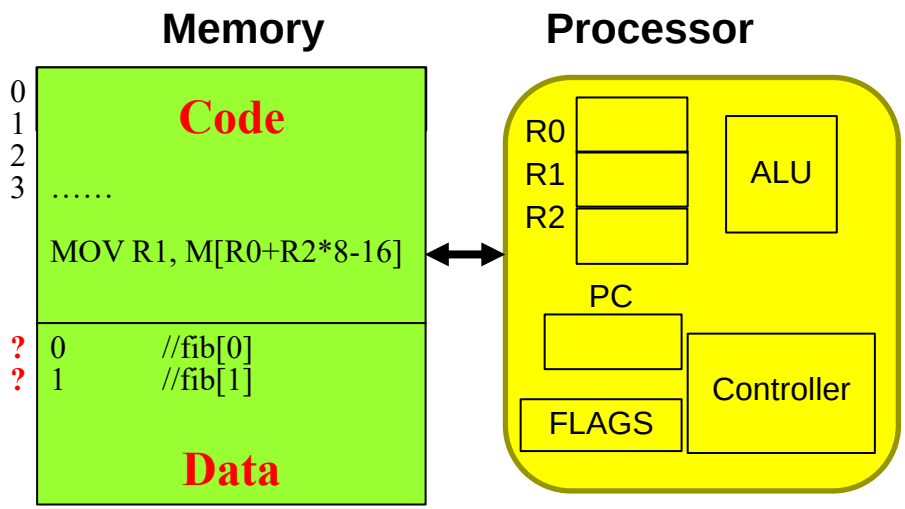
FC缺点：当R2==51，指令JL不会执行Loop→PC操作；
不转移到Loop，而是执行PC+2→PC操作，执行顺序下一条指令（在地址24）

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 初始值=24 R1: 累加器 R2: 索引寄存器 地址= 基址+索引*8+偏移量 Address=base+index*8+offset fib[i-2]所在地址 =R0+R2*8 -16 fib[i-1]所在地址 =R0+R2*8 -8 fib[i]所在地址 =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=12 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 初始值=? R1: 累加器 R2: 索引寄存器 地址= 基址+索引*8+偏移量 Address=base+index*8+offset fib[i-2]所在地址 =R0+R2*8 -16 fib[i-1]所在地址 =R0+R2*8 -8 fib[i]所在地址 =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://www.solid.things.ac.cn:7245/basic_programming