

# Lab3

余俊洁<sup>[523030910244]</sup>

YuJunjie-yjj@sjtu.edu.cn

## 1 Control Logic

- 取址 (Fetch):
  - 从当前 PC 所指内存中读取指令;
  - PC 自增更新, 准备下一条指令;
  - 将指令转为二进制数组 `command[16]` 为后续译码做准备
- 译码 (Decode):
  - 将 `command[16]` 进行解码得到相应的 opcode.
- 执行 (Excute):

Operation	Opcode	Functions
ADD(2 ways)	0001 (1)	ADD()
AND(2 ways)	0101 (5)	AND()
BR	0000 (0)	BR()
JMP	1100 (12)	JMP()
JSR	0100 (4)	JSR()
LD	0010 (2)	LD()
LDI	1010 (10)	LDI()
LDR	0110 (6)	LDR()
LEA	1110 (14)	LEA()
NOT	1001 (9)	NOT()
RET (achieved by JMP)	1100 (12)	RET()
RTI (not requested)	1000 (8)	RTI()
ST	0011 (3)	ST()
STI	1011 (11)	STI()
STR	0111 (7)	STR()
TRAP	1111 (15)	TRAP()

- 使用 switch-case 结构分发到不同的执行函数
- 进行状态更新

## 2 Important Functions

### 2.1 辅助函数

1. `bin_to_dec()`: 将一个位于 `num[]` 数组中、从 `begin` 到 `end` 的二进制位字段转换为一个十进制整数（支持补码）
2. `dec_to_bin()`: 将一个十进制整数 `n` 转换成一个长度为 16 的二进制数组 `res[]`，用于位级逻辑操作
3. `setNZP()`: 根据寄存器或操作结果 `num` 的值，更新 LC-3 的条件码 (N/Z/P) 的值

### 2.2 操作函数: `ADD()`, `BR()`, `LD()`, ...

## 3 Verification

为了检验各函数操作的正确性以及方便 debug，我写了一份包含了所有指令的一个汇编程序 ‘example.asm’

```

.ORG x3000

    LEA R6, DATA      ; LEA
    LD R0, NUM1        ; LD
    LDI R1, NUM1_PTR   ; LDI
    LDR R2, R6, #0     ; LDR (R6 指向 DATA)
    ST R0, SAVE1       ; ST
    STI R1, SAVE2_PTR  ; STI
    STR R2, R6, #1     ; STR

    ADD R3, R0, #5     ; ADD
    AND R4, R1, R2     ; AND
    NOT R5, R3         ; NOT

    BRzp SKIP         ; BR
    JSR SUBROUTINE     ; JSR10
    ADD R6, R6, #-2
SKIP JMP R6            ; JMP

    TRAP x25          ; HALT

SUBROUTINE|
    ADD R0, R0, #1
    RET

DATA .FILL xABCD      ; x3011
NUM1 .FILL x0005
NUM1_PTR .FILL NUM1
SAVE1 .BLKW 1
SAVE2_PTR .FILL SAVE2
SAVE2 .BLKW 1

.END

```

图 1. example.asm

同时，为了验证正确性，我不仅会直接根据 example.asm 分析运行的结果进行验证，还将通过将我的 simulator 运行的结果与我在 windows 主机上安装官方的 simulator 在各个断点运行的结果相对比。

```

augety@connet:~/Lab03$ ./simulate isaprogram
LC-3 Simulator

Read 23 words from program into memory.

LC-3-SIM> run 4

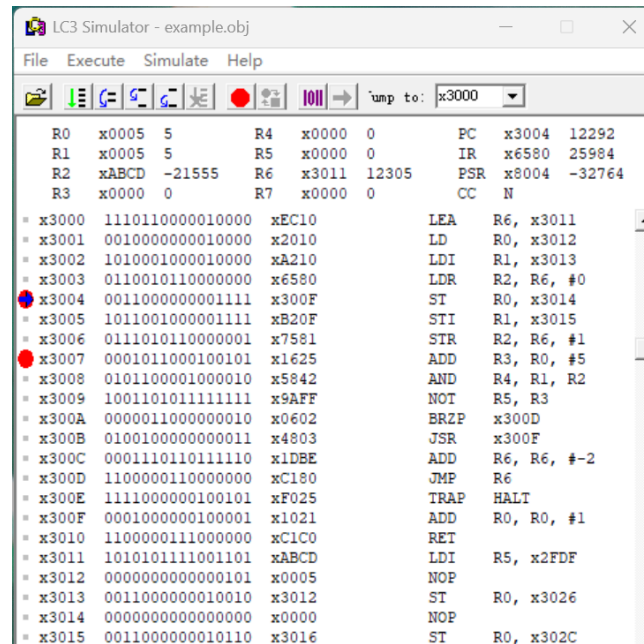
Simulating for 4 cycles...

LC-3-SIM> rdump

Current register/bus values :
-----
Instruction Count : 4
PC      : 0x3004
CCs: N = 1  Z = 0  P = 0
Registers:
0: 0x0005
1: 0x0005
2: 0xabcd
3: 0x0000
4: 0x0000
5: 0x0000
6: 0x3011
7: 0x0000
LC-3-SIM>

```

图 2. LD 系列操作



Address	Hex	Binary	Instruction
* x3000	1110110000010000	xECl0	LEA R6, x3011
* x3001	0010000000010000	x2010	LD R0, x3012
* x3002	1010001000010000	xA210	LDI R1, x3013
* x3003	0110010110000000	x6580	LDR R2, R6, #0
* x3004	0011000000001111	x300F	ST R0, x3014
* x3005	1011001000001111	xB20F	STI R1, x3015
* x3006	0111010110000001	x7581	STR R2, R6, #1
* x3007	0001011000100101	x1625	ADD R3, R0, #5
* x3008	0101100001000010	x5842	AND R4, R1, R2
* x3009	1001101011111111	x9AFF	NOT R5, R3
* x300A	0000011000000010	x0602	BRZP x300D
* x300B	0100100000000011	x4803	JSR x300F
* x300C	0001110110111110	x1DBE	ADD R6, R6, #-2
* x300D	1100000110000000	xCl80	JMP R6
* x300E	1111000000100101	xF025	TRAP HALT
* x300F	0001000000100001	x1021	ADD R0, R0, #1
* x3010	1100000111000000	xC1C0	RET
* x3011	1010101111001101	xABCD	LDI R5, x2FDF
* x3012	0000000000000101	x0005	NOP
* x3013	0011000000010010	x3012	ST R0, x3026
* x3014	0000000000000000	x0000	NOP
* x3015	0011000000010110	x3016	ST R0, x302C

图 3. LD 系列操作

在执行 x3000 到 x3003 的四条不同的 LD 指令之后，我的 simulator 和官方的得到的各个寄存器的值保持一致，证明了我的 simulator 对于 LEA,LD,LDI,LDR 指令实现的正确性。

具体来说，通过 LEA 指令，我们将 DATA 标签的地址也就是 x3011 存进了 R6；通过 LD 指令，将 NUM1 的值也就是 x0005 存进 R0；通过 LDI 指令，将 NUM1\_PTR 指向的值 (NUM1) 作为地址对应的值 x0005 存入 R1；通过 LDR 指令，将 R6 存的值作为地址 (x3011) 指向的值 xABCD 存入 R2。经检验，得到的结果均符合我们的分析。

```
LC-3-SIM> run 3
Simulating for 3 cycles...
LC-3-SIM> mdump 0x3011 0x3016

Memory content [0x3011..0x3016] :
-----
0x3011 (12305) : 0xabcd
0x3012 (12306) : 0xabcd
0x3013 (12307) : 0x3012
0x3014 (12308) : 0x05
0x3015 (12309) : 0x3016
0x3016 (12310) : 0x05
LC-3-SIM> 
```

图 4. ST 系列操作

x3011	1010101111001101	xABCD	LDI	R5, x2FDF
x3012	1010101111001101	xABCD	LDI	R5, x2FE0
x3013	0011000000010010	x3012	ST	R0, x3026
x3014	0000000000000101	x0005	NOP	
x3015	0011000000010110	x3016	ST	R0, x302C
x3016	0000000000000101	x0005	NOP	

图 5. ST 系列操作

在执行 x3004 到 x3006 的三条不同的 ST 指令之后，我的 simulator 和官方的得到的对应的地址内存入的值保持一致，证明了我的 simulator 对于 ST,STI,STR 指令实现的正确性。

具体来说，通过 ST 指令，我们将 R0 的值存在了 SAVE1 对应的地址 (x3010)；通过 STI 指令，我们将 R1 的值存在了 SAVE2\_PTR 对应的值对应的地址，也即是 SAVE2 的位置 (x3016)；通过 STR 指令，我们将 R2 的值，存在了 R6 的值 +1 对应的地址，也即 NUM1 的位置 (x3012)。经检验，得到的结果均符合我们的分析。



```

augety@comnet: ~/lab03
File Edit View Search Terminal Help

LC-3-SIM> run 2
Simulating for 2 cycles...
LC-3-SIM> rdump

Current register/bus values :
-----
Instruction Count : 12
PC : 0x300f
CCs: N = 1 Z = 0 P = 0
Registers:
0: 0x0005
1: 0x0005
2: 0xabcd
3: 0x000a
4: 0x0005
5: 0xffff5
6: 0x3011
7: 0x300c

LC-3-SIM> run 2
Simulating for 2 cycles...
LC-3-SIM> rdump

Current register/bus values :
-----
Instruction Count : 14
PC : 0x300c
CCs: N = 0 Z = 0 P = 1
Registers:
0: 0x0005
1: 0x0005
2: 0xabcd
3: 0x000a
4: 0x0005
5: 0xffff5
6: 0x3011
7: 0x3011

LC-3-SIM>

```

图 8. JSR 操作

```

LC3 Simulator - example.obj
File Execute Simulate Help

[Icons] [Dump to: x3000]

R0 x0006 6      R4 x0005 5      PC x300C 12300
R1 x0005 5      R5 xFFFF5 -11    IR xC1C0 -15936
R2 xABCD -21555 R6 x3011 12305   FSR x8001 -32767
R3 x000A 10     R7 x300C 12300   CC P

* x3000 1110110000010000 xEC10 LEA R6, x3011
* x3001 0010000000010000 x2010 LD R0, x3012
* x3002 1010001000010000 xA210 LDI R1, x3013
* x3003 0110010110000000 x6580 LDR R2, R6, #0
* x3004 0011000000001111 x300F STI R0, x3014
* x3005 1011001000001111 xB20F STI R1, x3015
* x3006 0111010110000001 x7581 STR R2, R6, #1
* x3007 0001011000100101 x1625 ADD R3, R0, #5
* x3008 0101100001000010 x5842 AND R4, R1, R2
* x3009 1001101011111111 x9AFF NOT R5, R3
* x300A 0000001100000010 x0602 BRZP x300D
* x300B 0100100000000011 x4803 JSR x300F
* x300C 0001110110111110 x1DBE ADD R6, R6, #-2
* x300D 1100000110000000 xC180 JMP R6
* x300E 1111000000100101 xF025 TRAP HALT
* x300F 0001000000100001 x1021 ADD R0, R0, #1
* x3010 1100000111000000 xC1C0 RET
* x3011 1010101111001101 xABCD LDI R5, x2FDF
* x3012 1010101111001101 xABCD LDI R5, x2FE0
* x3013 0011000000010010 x3012 ST R0, x3026
* x3014 000000000000101 x0005 NOP
* x3015 0011000000010110 x3016 ST R0, x302C

```

图 9. JSR 操作

由于执行完 NOT 指令,  $N = 1$ , 故 BR 指令不执行跳转, 此处 simulator 也确实没有跳转, 可得对于 BR 指令的正确性; 接着执行 JSR 指令跳转到 SUBROUTINE, 执行完  $R0+1$  的操作之后返回原程序。故此时  $PC=0x300c$ ,  $R0 = x0006$ 。得到的结果与分析一致, 也与官方的 simulator 一致。可得对于 JSR 指令的正确性。

```

augety@comnet: ~/lab03
File Edit View Search Terminal Help
LC-3-SIM> run 2
Simulating for 2 cycles...
LC-3-SIM> rdump

Current register/bus values :
-----
Instruction Count : 16
PC                : 0x300f
CCs: N = 0  Z = 0  P = 1
Registers:
0: 0x0000
1: 0x0005
2: 0xabcd
3: 0x000a
4: 0x0005
5: 0xffff
6: 0x300f
7: 0x300e
LC-3-SIM> run 2

```

图 10. JMP 操作

```

augety@comnet: ~/lab03
File Edit View Search Terminal Help
LC-3-SIM> run 2
Simulating for 2 cycles...
LC-3-SIM> rdump

Current register/bus values :
-----
Instruction Count : 18
PC                : 0x300e
CCs: N = 0  Z = 0  P = 1
Registers:
0: 0x0007
1: 0x0005
2: 0xabcd
3: 0x000a
4: 0x0005
5: 0xffff
6: 0x300f
7: 0x3011

LC-3-SIM> run 1
Simulating for 1 cycles...
LC-3-SIM> rdump

Current register/bus values :
-----
Instruction Count : 19
PC                : 0x0000
CCs: N = 0  Z = 0  P = 1
Registers:
0: 0x0007
1: 0x0005
2: 0xabcd
3: 0x000a
4: 0x0005
5: 0xffff
6: 0x300f
7: 0x3011

LC-3-SIM> run 1
Simulating for 1 cycles...
Simulator halted
LC-3-SIM>

```

图 11. TRAP 操作

当我们再 run 两步执行 JMP 指令，再一次跳到 SUBROUTINE 子进程，此时 PC 的值为 x300F, 与得到的结果一致，当我们再执行两步之后，此时的 PC 的值为 x300E, 又跳回了原程序并将要执行 TRAP x25 指令终止程序，符合实验结果。当我们再执行一步，即为执行 TRAP x25 指令，此时按照题目要求，我们需要将 PC 置为 0，进而终止程序运行，也与我们的实验结果想吻合。由此可得 simulator 对于 JMP 指令和 TRAP 指令的正确性。

由此，通过对于每一个操作指令的验证之后，我们可以验证我所编写的 simulator 的完备性与正确性。