Lab3 Documentation

SUN Yilin 520030910361

January 2, 2022

1 the Overall Control Logic

The idea is that the process_instruction function simulates the whole cycle of LC3, i.e. FETCH, DECODE, EVALUATE ADDRESS, FETCH OPERAND, EXECUTE, STORE RESULT.
Actually to implement the functions of the LC3 simulator we do not have to specify the six phases, we can simply integrate them together and the program will still work well. But for a better program structure and better code readability I will still divide the code into these phases.
Below is the description of what the program does in each phase and the reason why it does so.

1.1 FETCH

In this phase the simulator does two things. First it takes an instruction from the array MEMORY, then it increment the CURRENT_LATCHES.PC.

1.2 DECODE

This phase only calculates the opcode by right-shifting the instruction we have just derived from the previous phase.

1.3 EVALUATE ADDRESS

This phase does a lot of calculations to prepare all needed address, offsets and registers to be used for the upcoming phases. All these needed values are derived from the instruction. To be detail, we calculate PCoffset6, PCoffset9 and PCoffset11. We calculate the Trapvector, Destination Registers and Source Registers. We also calculate the BaseRoffset6.

1.4 FETCH OPERAND

This phase fetch all the needed operands. To be detail, the immediate value is directly derived from the instruction. And the values stored in MEMORY is fetched by the addresses we have calculated from the previous phase.

1.5 EXECUTE and STORE RESULT

Firstly we combine the two phases together because we are not using actual integrated circuits, hence executing and storing at the same time will be much easier for our program.
This is the most important part of our whole program. In the previous phases basically we are just preparing all the values needed but not doing actual operations. But here is our core function of the LC3 simulator. Our program uses a switch logic to deal with different instructions according to the opcode we have calculated in the DECODE phase. In the next section I will describe how the program does this in detail.

2 Important Functions in the Implementation

In this section I will briefly describe the important details in the implementation.

2.1 Get Offset Values and Addresses from the Instruction

We have already get the current Instruction from the Memory. To get the PC-offsets, BaseR-offsets and addresses, basically we only need to right-shift the Instruction by certain bits accordingly.
However, we need to pay attention when we are calculating the PC-offset values. Our values of the current PC is a 16-digit integer which is always positive, while our PC-offset calculated just now is a 32-digit integer as required by a standard C program. If the offset value is positive, we can directly add PC with it without having to worry about the difference in their digits. But when the offset value is negative, we must adjust it by text-extending it to get a 32-digit integer.

2.2 Set Condition Codes

To set condition codes efficiently, I use a function set_CC which takes an integer as input and set the condition codes accordingly. The input is the result during the execution of our program.

2.3 Core Function: EXECUTE and STORE RESULT

The core function is implemented by a switch logic.
Basically, the switch logic takes the opcode and do the simulation accordingly. For some specific operations such as AND, ADD and JSR, we need to pay attention to their different addressing modes. We need to decide whether the AND and ADD instruction are using immediate value or not. We also need to decide whether the JSR instruction is using PC-offset or BaseR directly.
For the BR instruction, we may directly get the nzp values from the instruction then compare them with the values stored in CURRENT_LATCHES.
For the LD, LDI, LDR and ST, STI, STR instructions, we have already prepared the values for them in the previous FETCH OPERAND stage and denote them by MDRd, MDRi and MDRr respectively. Now we only need to load the proper value into the Registers or MEMORY.

3 Verification

In this section I will use an assemble program to test various functions of the simulator and report the result.

3.1 the Assemble Program for Verification

Below is the full program I used to verify the correctness of the simulator. This program contains all instructions in the LC3 ISA.(Although it may not use all addressing modes for a single instruction, for example ADD) In the next subsection I will run this program step by step and show the state of Registers and Memory of the simulator.

```
; initialization

.ORIG x2000
AND R0,R0,x0
AND R1,R1,x0

; check ADD,AND,NOT and BR

ADD R0,R0,x-7
ADD R1,R1,x4
LOOP ADD R0,R1,R0
BRn LOOP
AND R1,R1,R0
NOT R1,R1

; check the Data Movement Instructions

LEA R3,x20
LEA R4,x20
ST R3,x-4
STI R4,x-5
LDI R5,x-6
LD R6,x-7
LDR R0,R6,x0
STR R0,R5,x1

; check JSR, JMP(by RET) and TRAP

AND R1,R1,x0
LEA R2,x2
JSRR R2
HALT
ADD R1,R1,x7
RET
.END
```

3.2 Step-by-step Running Result

To show the correctness of the program, I run this program step by step and report the result after each step. In the upcoming pages I attached the result from the file dumpsim and show the state after each line of the assemble program.
I have checked that the state of Registers and Memory are correct after each line of assmeble program. Please kindly read the following pages for more details.

```
.ORIG x2000
------------------------------------
Instruction Count : 0
PC                : 0x2000
CCs: N = 0  Z = 1  P = 0
Registers:
0: 0x0000
1: 0x0000
2: 0x0000
3: 0x0000
4: 0x0000
5: 0x0000
6: 0x0000
7: 0x0000


AND R0,R0,x0
------------------------------------
Instruction Count : 1
PC                : 0x2001
CCs: N = 0  Z = 1  P = 0
Registers:
0: 0x0000
1: 0x0000
2: 0x0000
3: 0x0000
4: 0x0000
5: 0x0000
6: 0x0000
7: 0x0000


AND R1,R1,x0
------------------------------------
Instruction Count : 2
PC                : 0x2002
CCs: N = 0  Z = 1  P = 0
Registers:
0: 0x0000
1: 0x0000
2: 0x0000
3: 0x0000
4: 0x0000
5: 0x0000
6: 0x0000
7: 0x0000


ADD R0,R0,x-7
------------------------------------
Instruction Count : 3
PC                : 0x2003
CCs: N = 1  Z = 0  P = 0
Registers:
```

```
0: 0xfff9
1: 0x0000
2: 0x0000
3: 0x0000
4: 0x0000
5: 0x0000
6: 0x0000
7: 0x0000


ADD R1,R1,x4
-------------------------------------
Instruction Count : 4
PC                : 0x2004
CCs: N = 0  Z = 0  P = 1
Registers:
0: 0xfff9
1: 0x0004
2: 0x0000
3: 0x0000
4: 0x0000
5: 0x0000
6: 0x0000
7: 0x0000


LOOP ADD R0,R1,R0
-------------------------------------
Instruction Count : 5
PC                : 0x2005
CCs: N = 1  Z = 0  P = 0
Registers:
0: 0xfffd
1: 0x0004
2: 0x0000
3: 0x0000
4: 0x0000
5: 0x0000
6: 0x0000
7: 0x0000


BRn LOOP
-------------------------------------
Instruction Count : 6
PC                : 0x2004
CCs: N = 1  Z = 0  P = 0
Registers:
0: 0xfffd
1: 0x0004
2: 0x0000
3: 0x0000
4: 0x0000
5: 0x0000
```

```
6: 0x0000
7: 0x0000


LOOP ADD R0,R1,R0
-------------------------------------
Instruction Count : 7
PC                : 0x2005
CCs: N = 0  Z = 0  P = 1
Registers:
0: 0x0001
1: 0x0004
2: 0x0000
3: 0x0000
4: 0x0000
5: 0x0000
6: 0x0000
7: 0x0000


BRn LOOP
-------------------------------------
Instruction Count : 8
PC                : 0x2006
CCs: N = 0  Z = 0  P = 1
Registers:
0: 0x0001
1: 0x0004
2: 0x0000
3: 0x0000
4: 0x0000
5: 0x0000
6: 0x0000
7: 0x0000


AND R1,R1,R0
-------------------------------------
Instruction Count : 9
PC                : 0x2007
CCs: N = 0  Z = 1  P = 0
Registers:
0: 0x0001
1: 0x0000
2: 0x0000
3: 0x0000
4: 0x0000
5: 0x0000
6: 0x0000
7: 0x0000


NOT R1,R1
-------------------------------------
```

```
Instruction Count : 10
PC                : 0x2008
CCs: N = 1  Z = 0  P = 0
Registers:
0: 0x0001
1: 0xffff
2: 0x0000
3: 0x0000
4: 0x0000
5: 0x0000
6: 0x0000
7: 0x0000


LEA R3,x20
-----------------------------------
Instruction Count : 11
PC                : 0x2009
CCs: N = 1  Z = 0  P = 0
Registers:
0: 0x0001
1: 0xffff
2: 0x0000
3: 0x2029
4: 0x0000
5: 0x0000
6: 0x0000
7: 0x0000


LEA R4,x20
-----------------------------------
Instruction Count : 12
PC                : 0x200a
CCs: N = 1  Z = 0  P = 0
Registers:
0: 0x0001
1: 0xffff
2: 0x0000
3: 0x2029
4: 0x202a
5: 0x0000
6: 0x0000
7: 0x0000


ST R3,x-4
Memory content [0x2000..0x2020] :
-----------------------------------
 0x2000 (8192) : 0x5020
 0x2001 (8193) : 0x5260
 0x2002 (8194) : 0x1039
 0x2003 (8195) : 0x1264
 0x2004 (8196) : 0x1040
```

```
 0x2005 (8197) : 0x9fe
 0x2006 (8198) : 0x5240
 0x2007 (8199) : 0x2029
 0x2008 (8200) : 0xe620
 0x2009 (8201) : 0xe820
 0x200a (8202) : 0x37fc
 0x200b (8203) : 0xb9fb
 0x200c (8204) : 0xabfa
 0x200d (8205) : 0x2df9
 0x200e (8206) : 0x6180
 0x200f (8207) : 0x7141
 0x2010 (8208) : 0x5260
 0x2011 (8209) : 0xe402
 0x2012 (8210) : 0x4080
 0x2013 (8211) : 0xf025
 0x2014 (8212) : 0x1267
 0x2015 (8213) : 0xc1c0
 0x2016 (8214) : 0x00
 0x2017 (8215) : 0x00
 0x2018 (8216) : 0x00
 0x2019 (8217) : 0x00
 0x201a (8218) : 0x00
 0x201b (8219) : 0x00
 0x201c (8220) : 0x00
 0x201d (8221) : 0x00
 0x201e (8222) : 0x00
 0x201f (8223) : 0x00
 0x2020 (8224) : 0x00


STI R4,x-5
-----------------------------------
 0x2028 (8232) : 0x00
 0x2029 (8233) : 0x202a
 0x202a (8234) : 0x00
 0x202b (8235) : 0x00
 0x202c (8236) : 0x00


LDI R5,x-6
-----------------------------------
Instruction Count : 15
PC                : 0x200d
CCs: N = 0  Z = 0  P = 1
Registers:
0: 0x0001
1: 0xffff
2: 0x0000
3: 0x2029
4: 0x202a
5: 0x202a
6: 0x0000
7: 0x0000
```

```
LD R6,x-7
-------------------------------------
Instruction Count : 16
PC                : 0x200e
CCs: N = 0  Z = 0  P = 1
Registers:
0: 0x0001
1: 0xffff
2: 0x0000
3: 0x2029
4: 0x202a
5: 0x202a
6: 0x2029
7: 0x0000


LDR R0,R6,x0
-------------------------------------
Instruction Count : 17
PC                : 0x200f
CCs: N = 0  Z = 0  P = 1
Registers:
0: 0x202a
1: 0xffff
2: 0x0000
3: 0x2029
4: 0x202a
5: 0x202a
6: 0x2029
7: 0x0000


STR R0,R5,x1
-------------------------------------
 0x2028 (8232) : 0x00
 0x2029 (8233) : 0x202a
 0x202a (8234) : 0x00
 0x202b (8235) : 0x202a
 0x202c (8236) : 0x00


AND R1,R1,x0
-------------------------------------
Instruction Count : 19
PC                : 0x2011
CCs: N = 0  Z = 1  P = 0
Registers:
0: 0x202a
1: 0x0000
2: 0x0000
3: 0x2029
4: 0x202a
5: 0x202a
```

```
6: 0x2029
7: 0x0000


LEA R2,x2
-------------------------------------
Instruction Count : 20
PC                : 0x2012
CCs: N = 0  Z = 1  P = 0
Registers:
0: 0x202a
1: 0x0000
2: 0x2014
3: 0x2029
4: 0x202a
5: 0x202a
6: 0x2029
7: 0x0000


JSRR R2
-------------------------------------
Instruction Count : 21
PC                : 0x2014
CCs: N = 0  Z = 1  P = 0
Registers:
0: 0x202a
1: 0x0000
2: 0x2014
3: 0x2029
4: 0x202a
5: 0x202a
6: 0x2029
7: 0x2013


ADD R1,R1,x7
-------------------------------------
Instruction Count : 22
PC                : 0x2015
CCs: N = 0  Z = 0  P = 1
Registers:
0: 0x202a
1: 0x0007
2: 0x2014
3: 0x2029
4: 0x202a
5: 0x202a
6: 0x2029
7: 0x2013


RET
-------------------------------------
```

```
Instruction Count : 23
PC                : 0x2013
CCs: N = 0  Z = 0  P = 1
Registers:
0: 0x202a
1: 0x0007
2: 0x2014
3: 0x2029
4: 0x202a
5: 0x202a
6: 0x2029
7: 0x2013


HALT
------------------------------------
Instruction Count : 24
PC                : 0x0000
CCs: N = 0  Z = 0  P = 1
Registers:
0: 0x202a
1: 0x0007
2: 0x2014
3: 0x2029
4: 0x202a
5: 0x202a
6: 0x2029
7: 0x2013
```