Homework 5

> 2023.07.17 沈韵沨 3200104392

Chap 5

5.37 Using the overall data path in Figure 5.18, identify the elements that implement the STI instruction of Figure 5.8.

Elements implement the STI instruction are:

- IR, RegFile, SEXT(with IR[8:0])
- PC, ADDR2MUX, ADDR1MUX(set to PC), Adder(connected with the former 2 components)
- MARMUX, GateMARMAX, MAR
- Memory, MDR

5.39 Using the overall data path in Figure 5.18, identify the elements that implement the LEA instruction of Figure 5.6.

Elements implement the LEA instruction are:

- IR, Reg File, SEXT(with IR[8:0])
- PC, ADDR2MUX, ADDR1MUX(set to PC), Adder(connected with the former 2 components)
- MARMUX, GateMARMAX, MAR

Chap 6

6.24 A student is debugging his program. His program does not have access to memory locations x0000 to x2FFF. Why that is the case we will discuss before the end of the book. The term is "privileged memory" but not something for you to worry about today.

He sets a breakpoint at x3050, and then starts executing the program. When the program stops, he examines the contents of several memory locations and registers, then hits single step. The simulator executes one instruction and then stops. He again examines the contents of the memory locations and registers. They are as follows:

$x + [-32, +31] = x4201$

$x \in [x4170, x4233]$

	Before	After
PC	x3050	x3051
R0	x2F5F	xFFFF
R1	x4200	x4200
R2	x0123	x0123
R3	x2323	x2323
R4	x0010	x0010
R5	x0000	x0000
R6	x1000	x1000
R7	x0522	x0522
M[x3050]	**x6???**	**x6???**
M[x4200]	x5555	x5555
M[x4201]	xFFFF	xFFFF

Complete the contents of location x3050

Opcode				DR			BaseR			Offset 6					
0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	1
LDR				R0			R1			#1					

Chap 7

7.32 Consider the following semi-nonsense assembly language program:

```
line 1:             .ORIG x8003
line 2:             AND R1,R1,#0
line 3:             ADD R0,R1,#5
line 4:             ST  R1,B
line 5:             LD  R1,A
line 6:             BRz SKIP
line 7:             ST  R0,B
line 8:   SKIP      TRAP x25
line 9:   A         .BLKW #7
line 10:  B         .FILL #5
line 11:  BANNER .STRINGZ "We are done!"
line 12:  C         .FILL x0
line 13:            .END
```

A separate module will store a value in A before this program executes.

Construct the symbol table.

Show the result of assembly of lines 5 through 7 above. *Note:* the instruction at line 8 has already been assembled for you.

x8006																
x8007																
x8008																
x8009	1	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1

Note that two different things could cause location B to contain the value 5: the contents of line 7 or the contents of line 10. Explain the difference between line 7 causing the value 5 to be in location B and line 10 causing the value 5 to be in location B.

1. Construct the symbol table:

Symbol	Address
SKIP	x8009
A	x800A
B	x800B
BANNER	x800C
C	x800D

2. Show the result of assembly of line 5 through 7 above:

Address	Binary	Description
x8006	`0010 001 000000011`	LD R1, #3
x8007	`0000 010 000000001`	BRz #1
x8008	`0011 000 000000010`	ST R0, #2
x8009	`1111 0000 00100101`	[SKIP] Trap x25

3. Explain the difference between line 7 causing the value 5 to be location B and line 10 causing the value 5 to be in location B:
    - Line 7: *Fetch* data from register R0 first(which is value 5), then *store* the value to memory location B.
    - Line 10: Pseudo-Op `.FILL` *initialized* location B to the value #5.

7.34 It is often useful to find the midpoint between two values. For this problem, assume A and B are both even numbers, and A is less than B. For example, if A = 2 and B = 8, the midpoint is 5. The following program finds the midpoint of two even numbers A and B by continually incrementing the smaller number and decrementing the larger number. You can assume that A and B have been loaded with values before this program starts execution.
Your job: Insert the missing instructions.

```
          .ORIG   x3000
          LD      R0,A
          LD      R1,B
X         ------------------ (a)
          ------------------ (b)
          ADD     R2,R2,R1
          ------------------ (c)
          ADD     R1,R1,#-1
          ------------------ (d)
          BRnzp   X
DONE      ST      R1,C
          TRAP    x25
A         .BLKW   1
B         .BLKW   1
C         .BLKW   1
          .END
```

```
          .ORIG   x3000
          LD      R0,   A
          LD      R1,   B
X         NOT     R2,   R0            (a)
          ADD     R2,   R2,   #1      (b)
          ADD     R2,   R2,   R1
          BRz     DONE                (c)
          ADD     R1,   R1,   #-1
          ADD     R0,   R0,   #1      (d)
          BRnzp   X
DONE      ST      R1,   C
          TRAP    x25
A         .BLKW   1
B         .BLKW   1
C         .BLKW   1
          .END
```

```
R0 = A
R1 = B
R2 = NOT(R0) }
R2 += 1      }  R2 = (-R0)
R2 = R2+R1     => (-R0) + R1
if (R0 == R1) break => R2 == 0
R1 -= 1
R0 += 1
```