College of Charleston
Computer Science Department

doesn't need to be the most efficient

CSCI 350 Digital Logic and Computer Organization
G. Pothering
Spring 2020
Assignment 6 **Due April 9, 2020**

0. Using the register-transfer examples we studied in class as a guide, give the sequences of register-transfers, control signals, and needed to implement the following IJVM instructions. Note, these problems will not be graded since the solutions are in the book, however I expect to see them completed. They are here to prepare you for the problems in part 1.
   a. IAND
   b. ISTORE varnum
   c. IFEQ offset
   d. SWAP
   e. IRETURN

1. Consider a new instruction for our instruction set, the IADV instruction. Its format is as follows

   IADV ***val*** (for "add value"). Here ***val*** is a 16-bit value representing a signed constant.

   Meaning: The value currently on top of the stack is to be popped from the stack, ***val*** represented is to be added to this value, and the sum is to be pushed onto the stack.

   a. Assume the op-code for this instruction is **EE**. Since val is 16 bits, when stored in the "method area" of memory (see page 246), which is byte-addressable, we need to regard val as having the structure $\mathbf{val_1 val_0}$, where each val_i is one byte and val_1 constitutes the most significant byte.

      Show what this instruction would look like when stored in the method area. Your rendering of the stored instruction must show the op-code immediately followed by the bytes comprising **val**. Recall that the architecture we are using here is "big-endian" meaning that when a multi-byte value is stored in consecutive memory locations, the most significant byte occupies the byte with the lowest of the memory addresses used in storing the value.

   b. In your answer for part a. above, indicate using the symbol PC and an arrow, the byte being referenced by the program counter PC at the time the register transfers for IADV are to be performed.

   c. Give the sequence of register transfers needed to implement the IADV instruction. In formulating your answer you are going to have to reconstruct val from two bytes in memory to a 32 bit value in the data path. Our discussion of GOTO may be helpful here.

2. Consider another new instruction for our instruction set, the ISTIND instruction. Its format is as follows

   ISTIND ***varnum*** (for store indirect).

   Here, as with the ISTORE instruction, ***varnum*** is a 1-byte unsigned integer representing an index into the local variable region of a method. This local variable region is, of course, accessible via address stored in the register LV.

   Meaning: Unlike ISTORE we are not going to pop a word from the stack and store it in the variable indexed by ***varnum***. Instead we are going to use the value stored at this location as a *pointer*. That is, the value stored in the local variable gives the word address in memory where we want the va;ue popped from the stack to be stored.

   Give the sequence of register transfers needed to implement the ISTIND.

3. Consider yet another new instruction for our instruction set, the ILBD instruction (for load using base-displacement). Its format is as follows

ILBD ***offset***.

Here, as with the GOTO instruction, ***offset*** is a 2-byte signed integer.

Meaning: The signed 16-bit value represented by ***offset*** is to be added to the value currently at the top of the stack. This sum will represent a *memory address*, and the value stored at this address is to pushed onto the stack. Note the previous value that was on top of the stack is to remain on the stack; it is not to be popped from the stack.

Give the sequence of register transfers needed to implement the ILBD.

4. Using the register-transfer examples we studied in class as a guide, give sequences of register-transfers needed to implement the following instructions

   a. PMEM varnum – reads the value stored in the local variable region referenced by the 1-byte, unsigned intege varnum. If this stored value is negative then increment the value stored by 1, otherwise leave the value unchanged.

   b. VMEM varnum: decrements by 1 that value of the local variable referenced by varnum. As in other instruction where varnum is used, it is a 1-byte unsigned integer.

   c. TSET varnum -- The instruction reads the value stored in the local variable represented by varnum, and pushes the value on the stack. It also sets the value stored in the local variable referenced by ***varnum*** to 1.

Shetali Emmanuel

p.272

Register transfer	Control signals active
Fetch	
PC = [PC] + 1; fetch	1, fetch, pc, F0, F1, enb, inc
IADD	
MAR, SP = [SP]-1; read	4, read, sp, mar, F0, F1, enb, inva
H = [TOS]; MDR = M[MAR] at end of cycle	7, F1, enb, h
TOS, MDR = [H] + [MDR]; write	0, write F0, F1, ena, enb, tos, mdr
DUP	
MAR, SP = [SP]-1	4, sp, mar, F0, F1, enb, inva
MDR = [TOS], write	7,write,F1, enb, mdr
ILOAD	
H = [LV]	5, h, F1, enb
MAR = [H] + [MBR]u; read	3, mar, F0, F1, ena, enb, read
MAR, SP = [SP] + 1	4, mar, sp, F0, F1, enb, inc
PC = [PC] + 1; fetch; write	1, fetch, write, pc, F0, F1, enb, inc
TOS = [MDR]	0, tos, F1, enb
POP	
MAR, SP = [SP]-1; read	4, read, mar, sp, F0, F1, enb, inva, read
Wait for read to complete	read
TOS = [MDR]	0, tos, F1, enb
#C IAND	
MAR, SP = [SP] - 1; read	4, read, sp, mar, F0, F1, enb, inva
H = [TOS]; MDR = M[MAR] at end	7, F1, enb, h
TOS, MDR = [H] AND [MDR], write	0, tos, MDR, ena, enb

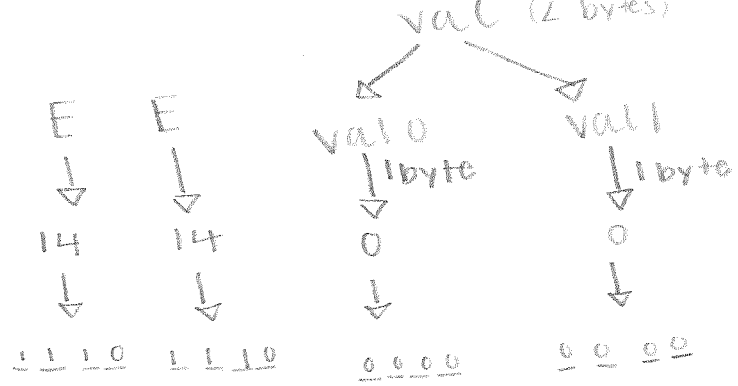
ISTORE varnum	
H = [LV]	5, h, F1, enb
MAR = [H] + [MBR]	3, mar, F0, F1, ena, enb
MDR = [TOS]; write	7, write, F1, enb, mdr
MAR, SP = [SP] - 1; read	4, mar, sp, F0, F1, enb, inva
PC = [PC] + 1; fetch	1, fetch, write, pc, F0, F1, enb, inc
TOS = [MDR]	0, tos, F1, enb
IFEQ offset	
MAR, SP = [SP] - 1; read	4, read, Sp, mar, F0, F1, enb, inva
H = [TOS]	7, h, F1, enb
TOS = [MDR]	0, tos, F1, enb
N = [OPC]; if (N) goto T else goto F	8, n, F1, enb
SWAP	
MAR = [SP] - 1; read	4, mar, F0, F1, enb, inva, read
MAR = [SP]	4, mar, F1, enb
H = [MDR]; write	0, h, F1, enb, write
MDR = [TOS]	7, mdr, F1, enb
MAR = [SP] - 1; write	4, mar, F0, F1, enb, inva, write
TOS = [H]	h, tos, F1, enb
IRETURN	
MAR, SP = [LV]; read	5, mar, Sp, F1, enb, read
wait for read to complete	read
LV, MAR = [MDR]; read	0, lv, mar, F1, enb, read
MAR = [LV] + 1	5, mar, F0, F1, enb, inc
PC = [MDR]; read; fetch	0, PC, F1, enb, read, fetch
MAR = [SP]	4, mar, F1, enb
LV = [MDR]	0, LV, F1, enb
MDR = [TOS]; write	7, mdr, F1, enb, write

pop a word

Store in local var

#1

[a]



[b]

points to the first & so PC is looking for the next instructions which are stored in the op code

[c] IADV

- pop value at the top of the Stack
- add val to value popped
- push sum onto stack

$H = [MBR]_S$

$PC = [PC] + 1$

$H = [H] >> 8$

$H = [H] + [MBR]_S$

$TOS = [H] + [TOS]$

#2

ISTIND varnum

└ use val stored @ varnum as a pointer

Store TOS at the address inside LV

(A) Get the address inside LV

H = [LV] + [MBR]

(B) get val @ TOS

MAR = [H]

(C) Store TOS @ address

MDR = [H] ; write

SP = [SP] - 1

pop

#3 ILBD

add offset + TOS

replace w/ TOS at SP+1

if negative put all 1's

(A) get true offset

H = [MBR]s >> 8

PC = [PC] + 1

H = [H] + [MBR]s

(B) add offset & TOS

MAR = [H] + [TOS]

SP = [SP] + 1

(C) push onto stack

TOS = [MDR] ; write

#4

[a] PMEM varnum

└ read val in LV

if <0 increment by 1

```
        H = [LV]
wait →  MAR = [H] + [MBR]; read
        N = [MDR]
        if (N<0) goto T else goto F
        T: LV = [MDR] + 1 ; write
```

[b] VMEM varnum

```
        H = LV
wait →  MAR = [H] + [MDR]; read
        MDR = [MDR] - 1 ; write
```

[c] TSET varnum

```
        H = [LV]
        MAR = [H] + [MBR] ; read
wait
        SP = [SP] + 1
        TOS = [MDR]
        MDR = 1 ; write
```