University of California, Berkeley
College of Engineering
Computer Science Division
Electrical Engineering and Computer Science Department

Computer Science 150 R. H. Katz
Fall 2000

Problem Set #10/11 (Assigned 16 November, Due 1 December)

1. Introduction

Your task is to complete the high level design of a very simple 4-bit computer. This will involve designing the datapath and a control state machine to the state diagram level to implement the instruction set. This is a very good example of the kind of comprehensive question you can expect to find on the final examination.

The machine has a single accumulator (R_0), a single bit carry register (C), four general purpose registers (R_0, R_1, R_2, R_3), four accumulator oriented instructions (COMP, INC, CLC, ADC), three register-register instructions (ADD, AND, XFER), two register-memory instructions (LOAD, STOR), and a conditional branch instruction (BRN). These instructions are encoded in from one to three 4-bit instruction parcels. The machine can address 256 x 4-bit words.

2. Instruction Set

Instructions are encoded in from one to three 4-bit words. Arithmetic instructions are encoded in a single word: two opcode bits (X_3X_2) and two operand bits (X_1X_0). All arithmetic instructions use R_0 as an accumulator. The C register receives the carry-out of all 4-bit additions. The operand bits specify one of the four on-chip registers. The COMP, INC, CLC and ADC instructions involve the Accumulator and Carry registers only, and use the operand bits to encode the four different operations.

Arithmetic Instructions

Op Code (Binary)	*Op Code (Symbolic)*	*Function*
00 X_1X_0	ADD	C, R[0] ← R[0] + R[X_1X_0]
01 X_1X_0	AND	R[0] ← R[0] AND R[X_1X_0]
10 00	COMP	R[0] ← ~R[0]
10 01	INC	C, R[0] ← R[0] + 1
10 10	CLC	C ← 0
10 11	ADC	C, R[0] ← R[0] + C

An opcode of 11 represents an extended instruction, and the remaining two bits of the word specify one of the four remaining instructions. An instruction word of 1100 encodes a register transfer instruction. The operands are encoded in the following word: the high order two bits identify the destination register, while the low order two bits identify the source.

Register Transfer Instruction

Op Code (Binary)	*Op Code (Symbolic)*	*Function*
1100 $Y_3Y_2Y_1Y_0$	XFER	R[Y_3Y_2] ← R[Y_1Y_0]

The remaining three instructions are encoded in three 4-bit words: 1 word for the extended op code, and two words to identify the target address (memory extends from location 0 through location 255_{10}).

Memory Reference Instructions

Op Code (Binary)	*Op Code (Symbolic)*	*Function*
1101 $Y_3Y_2Y_1Y_0$ $Z_3Z_2Z_1Z_0$	LOAD	R[0] ← MEM[$Y_3Y_2Y_1Y_0$ $Z_3Z_2Z_1Z_0$]
1110 $Y_3Y_2Y_1Y_0$ $Z_3Z_2Z_1Z_0$	STOR	MEM[$Y_3Y_2Y_1Y_0Z_3Z_2Z_1Z_0$] ← R[0]
1111 $Y_3Y_2Y_1Y_0$ $Z_3Z_2Z_1Z_0$	BRN	IF R[0]<3>=1 THEN PC ← $Y_3Y_2Y_1Y_0$ $Z_3Z_2Z_1Z_0$

3. Memory Interface

The processor has the following input/output connections to the outside world: RESET (on reset, the PC is set to ZERO), clock, bidirectional data lines $D_3D_2D_1D_0$, and address lines $A_7A_6A_5A_4A_3A_2A_1A_0$. The interface to memory is exactly as in Laboratory #5: the processor generates a $\overline{CS}$ (chip select) and $\overline{WE}$ (write enable) signal to control RAM directly. We make the significant simplifying assumption that the processor clock period will exceed the RAM access time (i.e., the memory is a 0-Wait State Memory). See Figure 1.

The processor should be implemented with a program ROM and a data RAM. Instructions are always fetched from the ROM.

To load/examine program data, the processor should also include a simple "operator's console" of status LEDs and input switches. It will include a GO push button and a READ/$\overline{WRITE}$ switch to load or examine the contents of RAM, along with a switch input to place the processor in a RUN/$\overline{EXAMINE}$ mode. When in RUN mode, the processor executes the normal instruction fetch/decode/execute cycles. The operator's console in inactive, and its memory interface signals are left unasserted. When placed in the $\overline{EXAMINE}$ mode, the processor completes the execution of the current instruction and enters an idle state, in which the address and data lines are tri-stated, and the memory interface signals are unasserted (driven high). The operator's console is now active. Eight external switch inputs determine the address of a memory location to be read and output to LEDs or written from four data switches. The READ/$\overline{WRITE}$ switch determines the direction of access. The GO button commences a RAM read or write cycle, including appropriate control for the latches and buffers of the console's datapath.

4. Sample Programs

The Carry Clear (CLC) and Add with Carry (ADC) instructions make it possible to implement additions of

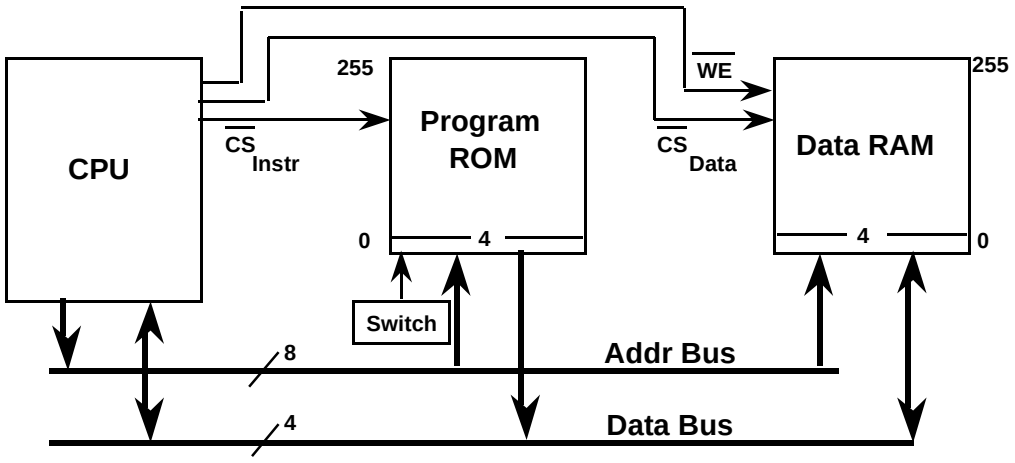

Figure 1: *CPU/Memory Interface*

greater than four bits. For example, the following sequence of instructions will perform an 8-bit addition of the values at memory locations W, X and Y, Z, and store the result into location U, V:

Address	Instruction	Comment
0	CLC	C ← 0
1	LOAD Z	R[0] ← Mem[Z]
4	XFER R1, R0	R[1] ← R[0]
6	LOAD X	R[0] ← Mem[X]
9	ADD R1	C, R[0] ← R[0] + R[1]
11	STOR V	Mem[V] ← R[0] (low order 4-bits of sum)
14	LOAD Y	R[0] ← Mem[Y]
17	ADC	C, R[0] ← R[0] + C (add carry from low order sum)
18	XFER R1, R0	R[1] ← R[0]
20	LOAD W	R[0] ← Mem[W]
23	ADD R1	C, R[0] ← R[0] + R[1]
24	STOR U	Mem[U] ← R[0]

Here is a sample program that sums the integers from 1 to 4:

Address	Instruction	Comment
0	LOAD #0	R[0] ← 0
3	R1 ← R0	R[1] (SUM) ← 0
5	LOAD #4	R[0] ← 4
8	COMP	R[0] ← ~4
9	INC	R[0] ← -4
10	R2 ← R0	R[2] (I) ← -4
12 X:	R0 ← R2	R[0] ← I
14	COMP	R[0] ← ~I
15	INC	R[0] ← -I
16	ADD R1	R[0] ← -I + SUM
17	R1 ← R0	SUM ← R[0]
19	R0 ← R2	R[0] ← I
21	INC	R[0] ← I + 1
22	R2 ← R0	I ← R[0]
24	BRN X	IF I < 0 GOTO X
27	R0 ← R1	R[0] ← SUM
29	STOR RES	RES ← R[0]
32	LOAD #0	R[0] ← 0
35	COMP	R[0] ← -1
36 Y:	BRN Y	LOOP FOREVER
253	RES	
254	#4	
255	#0	

5. What You Are Expected To Do

1. Design a detailed datapath for this machine, down to the level of registers, functional units, busses, multiplexers, and tri-state gates. Include the operator's console!
2. Create a list of all register transfer operations supported by your datapath and how these are implemented by detailed control signals.
3. Using your datapath for Part (1) and control signals for Part (2), write down a Moore or Synchronous Mealy Machine state diagram that implements the full instruction set, memory interface, and operator's console. The outputs from the State Machine can be the high-level register transfer operations rather than the detailed controlled signals.