ECE 375
Computer Organization and Assembly Language Programming
Winter 2013
Solutions Set #3

The following questions are based on the enhanced AVR datapath (see Figures 8.24 and 8.26 in the text). The microoperation for the Fetch cycle is shown below.

Stage	Micro-operations
IF	IR ← M[PC], PC ← PC + 1, NPC ← PC + 1, RAR ← PC + 1

[25 pts]

1- Consider the implementation of the `CPI Rd,K` (*Compare Register with Immediate*) instruction on the enhanced AVR datapath.
   (a) List and explain the sequence of microoperations required to implement `CPI Rd,K`.
   (b) List and explain the control signals and the Register Address Logic (RAL) output for the `CPI Rd,K` instruction.

   Note that this instruction takes one execute cycle (EX). Control signals for the Fetch cycle are given below. Clearly explain your reasoning.

Solution:

(a)

EX: Rd - K

(b)

Control Signals	IF	`CPI`
		EX
MJ	00	xx
MK	0	x
ML	0	x
IR_en	1	x
PC_en	1	0
PCh_en	0	0
PCl_en	0	0
NPC_en	1	x
SP_en	0	0
DEMUX	x	x
MA	x	0
MB	x	x
ALU_f	xxxx	0010
MC	xx	xx
RF_wA	0	0
RF_wB	0	0
MD	x	x
ME	x	x
DM_r	x	x
DM_w	0	0
MF	x	x
MG	xx	xx
Adder_f	xx	xx
MH	x	x
MI	x	x

RAL Output	`CPI`
	EX
wA	x
wB	x
rA	rd
rB	x

EX1:
The content of Rd is read from the Register File by providing Rd to rA. The immediate value K is routed to input-B of the ALU by setting MA to 0. Then, subtraction is performed but nothing is written back to the register file. The condition flags, N, Z, V, C, etc. will be set based on the outcome of the subtract operation. All other control signals can be "don't cares" except DM_w, SP_en, and PC_en, which all need to be set to 0's to prevent Data Memory, SP register, and PC register from being overwritten. Note that IR_en can be "don't care" since this is the last execute cycle and IR register will be overwritten in the Fetch (i.e., next) cycle.

[25 pts]

2- Consider the implementation of the `ST -X, Rr` (*Store Indirect and Pre-Decrement*) instruction on the enhanced AVR datapath.
   (a) List and explain the sequence of microoperations required to implement `ST -X, Rr`.
   (b) List and explain the control signals and the Register Address Logic (RAL) output for the `ST -X, Rr` instruction.
   Note that this instruction takes two execute cycles (EX1 and EX2). Control signals for the Fetch cycle are given below. Clearly explain your reasoning.

Solution:

(a)

EX1: DMAR ← Xh:XL - 1, Xh:Xl ← Xh:Xl - 1

EX2: M[DMAR]← Rr

(b)

RAL Output	`ST -X, Rr`	
	EX1	EX2
wA	x	x
wB	x	x
rA	Xh	x
rB	Xl	Rd

Control Signals	IF	`ST -X, Rr`	
		EX1	EX2
MJ	00	xx	xx
MK	0	x	x
ML	0	x	x
IR_en	1	0	x
PC_en	1	0	0
PCh_en	0	0	0
PCl_en	0	0	0
NPC_en	1	x	x
SP_en	0	0	0
DEMUX	x	x	x
MA	x	x	x
MB	x	x	x
ALU_f	xxxx	xxxx	xxxx
MC	xx	01	xx
RF_wA	0	1	0
RF_wB	0	1	0
MD	x	x	1
ME	x	x	1
DM_r	x	x	0
DM_w	0	0	1
MF	x	x	x
MG	xx	10	xx
Adder_f	xx	10	xx
MH	x	1	x
MI	x	x	x

EX1:
The contents of Xh and Xl are read from the Register File by providing Xh and Xl to rA and rB, respectively. Xh:Xl (or X) is routed to input-A of the Address Adder by setting MG to 10 and decrementing by 1. X-1 is routed through input-1 of MUXH by setting MH to 1, and latched onto DMAR as well as written to the Register file by setting both

RF_wA and RF_wB to 1's. All other control signals can be don't cares except IR_en, DM_w, PC_en, and SP_en, which needs to be set to 0 to prevent IR register, Data Memory, PC register, and SP register, respectively, from being overwritten.

EX2:
The address in DMAR is routed through MUXE by setting ME to 1 to provide address for the Data Memory. At the same time, the content of Rr is read from the register file by providing Rr to rB (note that Rr is the same as Rd for stores). Then, Rr is written to the Data Memory by setting DM_w to 1. All other control signals can be don't cares except PC_en and SP_en, which need to be set to 0 to prevent PC register and SP register, respectively, from being overwritten. Note that IR_en can be "don't care" since this is the last execute cycle and IR register will be overwritten in the fetch (i.e., next) cycle.

[25 pts]

3- Consider the implementation of the `ICALL` (Indirect Call to Subroutine) instruction on the enhanced AVR datapath shown below. `ICALL` is similar to the `RCALL` (Relative Call to Subroutine) instruction, except that the Z register points to the target address.
   (a) List and explain the sequence of microoperations required to implement `ICALL`.
   (b) List and explain the control signals and the Register Address Logic (RAL) output for the `ICALL` instruction
   Note that this instruction takes two execute cycles (EX1 and EX2). Control signals for the Fetch cycle are given below. Clearly explain your reasoning

Solution:

(a)

EX1: M[SP] ←RARl, SP ← SP - 1

EX2: M[SP] ←RARh, SP ← SP – 1, PC ← Z

(b)

Control Signals	IF	ICALL	
		EX1	EX2
MJ	00	xx	11
MK	0	x	x
ML	0	x	x
IR_en	1	0	x
PC_en	1	x	1
PCh_en	0	0	0
PCl_en	0	0	0
NPC_en	1	0	x
SP_en	0	1	1
DEMUX	x	x	x
MA	x	x	x
MB	x	x	x
ALU_f	xxxx	xxxx	xxxx
MC	xx	xx	xx
RF_wA	0	0	0
RF_wB	0	0	0
MD	x	0	0
ME	x	0	0
DM_r	x	0	0
DM_w	0	1	1
MF	x	x	x
MG	xx	00	00
Adder_f	xx	10	10
MH	x	x	0
MI	x	0	1

RAL Output	ICALL	
	EX1	EX2
wA	x	x
wB	x	x
rA	x	Zh
rB	x	Zl

EX1:
SP provides the address for the Data Memory by setting ME to 0, and then RARl is selected by setting MI to 0 and written to the Data Memory by setting MD to 0 and DM_w to 1. At the same, time, SP is decremented using the Address Adder by setting Adder_f to 10 and latched on to the SP by setting SP_en to 1. All other control signals can be "don't cares" except IR_en and SP_en, which needs to be set to 0 to prevent IR register and SP register, respectively, from being overwritten. Note that PC_en can be don't care since PC will be overwritten in EX1.

EX2:
SP provides the address for the Data Memory by setting ME to 0, and then RARh is selected by setting MI to 1 and written to the Data Memory by setting MD to 0 and DM_w to 1. SP is also decremented by the Address Adder by setting Adder_f to 10 and latched on to the SP by setting SP_en to 1. At the same time, Zh and Zl is read from the Register File and concatenated to form Z. Then, Z is routed to and latched onto the PC by setting MH to 0, MJ to 11, and PC_en to 1. All other control signals can be don't cares. Note that IR_en can be "don't care" since this is the last execute cycle and IR register will be overwritten in the fetch (i.e., next) cycle. Also note that NPC_en can be "don't care" because the content of RAR (as well as NPC) will not change until the end of the cycle.

[25 pts]

4- Consider the multi-cycle implementation of the `RET` (Return from subroutine) instruction on the enhanced AVR datapath.
   (a) List the sequence of microoperations required to implement `RET`.
   (b) List and explain the control signals and the Register Address Logic (RAL) output for the `RET` instruction.
   Note that this instruction takes three execute cycles (EX1, EX2, and EX3). The Fetch cycle is shown below.

Stage	Micro-operations
IF	IR ← M[PC], PC ← PC + 1, NPC ← PC + 1, RAR ← PC + 1

Solution:
(a)
EX1: SP ←SP +1
EX2: PCh ← M[SP], SP ← SP +1
EX3: PCl ← M[SP]

(b)

Control Signals	IF	RET		
		EX1	EX2	EX3
MJ	00	xx	xx	xx
MK	0	x	x	x
ML	0	x	x	x
IR_en	1	0	0	x
PC_en	1	0	0	0
PCh_en	0	0	1	0
PCl_en	0	0	0	1
NPC_en	1	x	x	x
SP_en	0	1	1	0
DEMUX	x	x	1	0
MA	x	x	x	x
MB	x	x	x	x

ALU_f	xxxx	xxxx	xxxx	xxxx
MC	x	x	x	x
RF_wA	0	0	0	0
RF_wB	0	0	0	0
MD	x	x	x	x
ME	x	x	0	0
DM_r	x	x	1	1
DM_w	0	0	0	0
MF	x	x	x	x
MG	00	00	00	xx
Adder_f	xx	01	01	xx
MH	x	x	x	x
MI	x	x	x	x

RAL Output	RET		
	EX1	EX2	EX3
wA	x	x	x
wB	x	x	x
rA	x	x	x
rB	x	x	x

EX1:
The content of SP is routed to input-A of the Address Adder by setting MG to 00, and incremented by setting Adder_f to 01. The incremented PC is then latched onto SP by setting Sp_en to 1. All other control signals can be "don't cares" except DM_w, IR_en, and PC_en, which all need to be set to 0's to prevent the Data Memory, IR, and PC being overwritten with unwanted values.

EX2:
The content of SP is routed to input-A of the Address Adder by setting MG to 00, and incremented by setting Adder_f to 01. The incremented PC is then latched onto SP by setting Sp_en to 1. At the same time, the Data Memory location pointed to by SP, i.e., M[SP], which is the higher byte of the return address, is read by providing SP as an address to the Data Memory by setting ME to 0 and DM_r to 1. The read value is then routed to DEMUX to the upper byte of PC, i.e., PCh by setting DEMUX to 1 and PCh_en to 1. All other control signals can be don't cares except DM_w, IR_en, and PC_en, which all need to be set to 0's to prevent the Data Memory, IR, and PC being overwritten with unwanted values.

EX3:
The Data Memory location pointed to by SP, i.e., M[SP], which is the lower byte of the return address, is read by providing SP as an address to the Data Memory by setting ME to 0 and DM_r to 1. The read value is then routed to DEMUX to the lower byte of PC, i.e., PCl, by setting DEMUX to 1 and PCl_en to 1. All other control signals can be don't cares except DM_w and PC_en, which all need to be set to 0's to prevent the Data Memory and PC being overwritten with unwanted values. Note that IR_en can be "don't care" since this is the last execute cycle and IR register will be overwritten in the Fetch (i.e., next) cycle.