ECE 375
Computer Organization and Assembly Language Programming
Fall 2022
Solutions Set #4

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 `PUSH Rr` (*Push Register on Stack*) instruction on the enhanced AVR datapath.
   (a) List and explain the sequence of microoperations required to implement `PUSH Rr`.
   (b) List and explain the control signals and the Register Address Logic (RAL) output for the `PUSH Rr` instruction.
   Note that this instruction takes one execute cycle (EX) (despite the fact that the datasheet indicates 2 execute cycles). Control signals for the Fetch cycle are given below. Clearly explain your reasoning

Control Signals	IF	`PUSH Rr`
		EX
MJ	0	x
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	1
DEMUX	x	x
MA	x	x
MB	x	x
ALU_f	xxxx	xxxx
MC	xx	xx
RF_wA	0	0
RF_wB	0	0
MD	x	1
ME	x	0
DM_r	x	0
DM_w	0	1
MF	x	x
MG	x	x
Adder_f	xx	xx
Inc_Dec	x	1
MH	x	x
MI	x	x

RAL Output	`PUSH Rr`
	EX1
wA	x
wB	x
rA	x
rB	Rr

Solution

(a) The microoperations required are given below:

EX: M[SP] ← Rr, SP ← SP - 1

(b) The control signals and RAL output requirements for the `PUSH Rr` instruction are shown below:

EX:
The content of the source register (Rr) is written to the Data Memory location pointed to by SP. This is done by setting MD to 1, ME to 0, DM_w to 1, and DM_r to 0. In addition, the SP is decremented and latched back to SP at the end of the cycle. This is done by setting Inc_Dec to 1 and setting SP_en to 1. All other control signals can be "don't cares" except for RF_wA and RF_wB which need to be set to 0 to prevent the Register File from being updated. In addition, IR_en, PC_en, PCh_en, PCl_en, which are all set to 0 to prevent IR and PC 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 `LD Rd,Y+` (*Load Indirect and Post-Increment*) instruction on the enhanced AVR datapath.

(a) List and explain the sequence of microoperations required to implement `LD Rd,Y+`.

(b) List and explain the control signals and the Register Address Logic (RAL) output for the `LD Rd,Y+` 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) Here is the sequence of microoperations for the `LD Rd,Y+` instruction.

EX1: DMAR ← Yh:YL, Yh:Yl ← Yh:Yl + 1
EX2: Rd ← M[DMAR]

(b) The following shows the control signals and the RAL output.

Control Signals	IF	`LD Rd, Y+` EX1	EX2
MJ	0	x	x
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	1
ALU_f	xxxx	xxxx	xxxx
MC	xx	01	00
RF_wA	0	1	0
RF_wB	0	1	1
MD	x	x	x
ME	x	x	1
DM_r	x	x	1
DM_w	0	0	0
MF	x	x	x

RAL Output	`LD Rd, Y+` EX1	EX2
wA	Yh	x
wB	Yl	Rd
rA	Yh	x
rB	Yl	x

MG	x	1	x
Adder_f	xx	01	xx
Inc_Dec	x	x	x
MH	x	0	x
MI	x	x	x

EX1:
The contents of Yh and Yl are read from the Register File by providing Yh and Yl to rA and rB, respectively. Yh:Yl or Y is routed to DMAR by setting MH to 0. At the same time, Y is incremented by one by the Address Adder (via MUXG) by setting Adder_f to 01, and then latched onto YH and YL (via MUXC) by setting both RF_wA and RF_wB to 1's and providing Yh and Yl to wA and wB, respectively. All other control signals can be don't cares except DM_w, which needs to be set to 0 so that the memory is not overwritten, and IR_en, PC_en, as well as PCh_en and PCl_en, and SP_en, which are all set to 0's to prevent IR, PC, and SP, respectively, from being overwritten. The RAL output for rA and rB are set to Yh and Yl, respectively, so that the upper and lower bytes of Y can be read from the register file. This is also the case for wA and wB since updated value of Y needs to be written back.

EX2:
The content of DMAR is routed through MUXE and used to fetch the operand from Data Memory. The fetched operand is routed through MUXB and MUXC to the inB of the register file and written by setting RF_wB to 1. All other control signals can be don't cares except DM_w, which needs to be set to 0 so that the memory is not overwritten. PC_en (as well as PCh_en and PCl_en), SP_en, RF_wA, which all need to be set to 0 to prevent the PC register, the SP register, and the register file, respectively, from being overwritten. Note that IR_en can be "don't care" since this is the last execute cycle and the IR register will be overwritten in the fetch (i.e., next) cycle. The RAL output for wB has to be set to Rd because the loaded value from memory has to be written to the destination register.

[25 pts]
3- Consider the implementation of the `RET` (*Return from subroutine*) instruction on the enhanced AVR datapath.
  (a) List and explain 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). Control signals for the Fetch cycle are given below. Clearly explain your reasoning.

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

(b)

Control Signals	IF	RET		
		EX1	EX2	EX3
MJ	0	x	x	x
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	xx	xx	xx	xx
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	x	x	x	x
Adder_f	xx	xx	xx	xx
Inc_Dec	x	0	0	x
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 the Increment/Decrement Unit, and incremented by setting Inc_Dec to 0. The incremented SP is then relatched onto SP by setting SP_en to 1. All other control signals can be "don't cares" except RF_wA/RF_wB, DM_w, IR_en, and PC_en (as well as PCh_en and PCl_en), which all need to be set to 0's to prevent the register file, Data Memory, IR, and PC from being overwritten with unwanted values.

EX2:
The content of SP is routed to the Increment/Decrement Unit, and incremented by setting Inc_Dec to 0. The incremented SP is then relatched 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 RF_wA/RF_wB, DM_w, IR_en, and PC_en, which all need to be set to 0's to prevent the register file, 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 0 and PCl_en to 1. All other control signals can be don't cares except Sp_en, RF_wA/RF_wB, DM_w and PC_en, which all need to be set to 0's to prevent the SP, register file, 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.

[25 pts]
4- Consider the implementation of the `LPM R16, Z+` (*Load Program Memory*) instruction on the enhanced AVR datapath.

(a) List and explain the sequence of microoperations required to implement `LPM R16, Z+`. Note that this instruction takes three execute cycles (EX1, EX2, and EX3).
(b) List and explain the control signals and the Register Address Logic (RAL) output for the `LPM` instruction.
Control signals for the Fetch cycle are given below. Clearly explain your reasoning.

Solution

(a) Here is the sequence of microoperations for the `LPM R16, Z+` instruction.

EX1: PMAR ← Zh:Zl
EX2: MDR ← M[PMAR], Z ← Z +1
EX3: R16 ← MDR

(b) The following shows the control signals and the RAL output.

Control Signals	IF	`LPM R16, Z+`		
		EX1	EX2	EX3
MJ	0	x	x	x
MK	0	x	x	x
ML	0	x	1	x
IR_en	1	0	0	x
PC_en	1	0	0	0
PCh_en	0	0	0	0
PCl_en	0	0	0	0
NPC_en	1	x	x	x
SP_en	0	0	0	0
DEMUX	x	x	x	x
MA	x	x	x	x
MB	x	x	x	x
ALU_f	xxxx	xxxx	xxxx	xxxx
MC	xx	xx	01	10
RF_wA	0	0	1	0
RF_wB	0	0	1	1
MD	x	x	x	x
ME	x	x	x	x
DM_r	x	x	x	x
DM_w	0	0	0	0
MF	x	x	x	x
MG	x	1	1	x
Adder_f	xx	11	01	xx
Inc_Dec	x	x	x	x
MH	x	x	x	x
MI	x	x	x	x

RAL Output	`LPM R16, Z+`		
	EX1	EX2	EX3
wA	x	Zh	x
wB	x	Zl	R16
rA	Zh	Zh	x
rB	Zl	Zl	x

EX1:
The contents of Zh and Zl registers are read from the Register File by providing Zh and Zl to rA and rB, respectively. Zh:Zl is then latched onto the PMAR register. This is done by routing through the Address Adder by setting MG to 1 and Adder_f to 11. All other control signals can be "don't cares" except RF_wA and RF_wB to prevent the register file from being updated. In addition, IR_en, DM_w, PC_en, PCh_en, PCl_en, and SP_en need to be set to 0's to prevent IR register, Data Memory, PC register, and SP register, respectively, from being overwritten. The RAL output for rA and rB are set to Zh and Zl, respectively, so that the upper and lower bytes of Z can be read from the register file.

EX2:
The Program Memory is read based on PMAR by setting ML to 1, and then the value read is latched onto MDR. In addition, Z is incremented. This is achieved by reading the Zh and Zl registers by providing Zh and Zl to rA and rB,

respectively. Zh:Zl or Z is incremented using the Address Adder by setting MG to 1 and Adder_f to 01. The incremented Z is writing back to the Register file by setting MC to 01 and providing Zh and Zl to wA and wB, respectively, and 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, PCh_en, PCl_en, and SP_en, which need to be set to 0's to prevent the IR register, Data Memory, PC register, and SP register, respectively, from being overwritten. Finally, the RAL output for rA and rB are set to Zh and Zl, respectively, so that the upper and lower bytes of Z can be read from the register file. Moreover, the RAL output for wA and wB are set to Zh and Zl, respectively, so that the upper and lower bytes of Z can be written back to the register file.

EX3:
The content of MDR is written back to R16 in the register file by setting MC to 10, wB to 0010000_2 (i.e., R16), and RF_wB to 1. All other control signals can be "don't cares", except RF_wA, DM_w, PC_en, PCh_en, PCl_en, and SP_en, which need to be set to 0's to prevent Register File, Data Memory, 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 the IR register will be overwritten in the Fetch (i.e., next) cycle. The RAL output for wB has to be set to Rd (which happens to be 0) because the loaded value from memory has to be written to a destination register.